Chapter 18 – Files and Streams

Outline

Slide modified by L. Lilien

Many slides modified by Prof. L. Lilien (even many without an explicit message).

18.1 Introduction

- Variables and arrays are only temporary
  - Lost during garbage collection or when a program terminates
- Files used for long term storage
  - Called persistent data
  - Stored on secondary storage devices
    - Can store large amounts of data
      - Much more than in variables or arrays in program memory
    - Can "carry" data between program executions
- This chapter deals with:
  - Sequential-access files
    - Ed. 1 only: Random-access files (not covered in CS 1120)
  - File processing features
  - Stream-input/output features

18.2 Data Hierarchy (see Fig – next page)

- Bit (binary digit) : either ‘1’ or ‘0’
  - All data represented as combination of bits
  - Easy for electronic devices to understand
- Byte: eight bits
- Character: two bytes in C# (C# uses Unicode)
  - Character set: set of all characters used to program and represent data on a particular computer
- Field: composition of characters that conveys a meaning
- Record: composition of several, related fields
- File: group of related records
  - Record key: identifies record (ties it to a particular entity)
  - Sequential file: typically, records stored in order of record keys (cf. slides 25-27)
- Database: a collection of related files (managed by DBMS)

18.2 Data Hierarchy

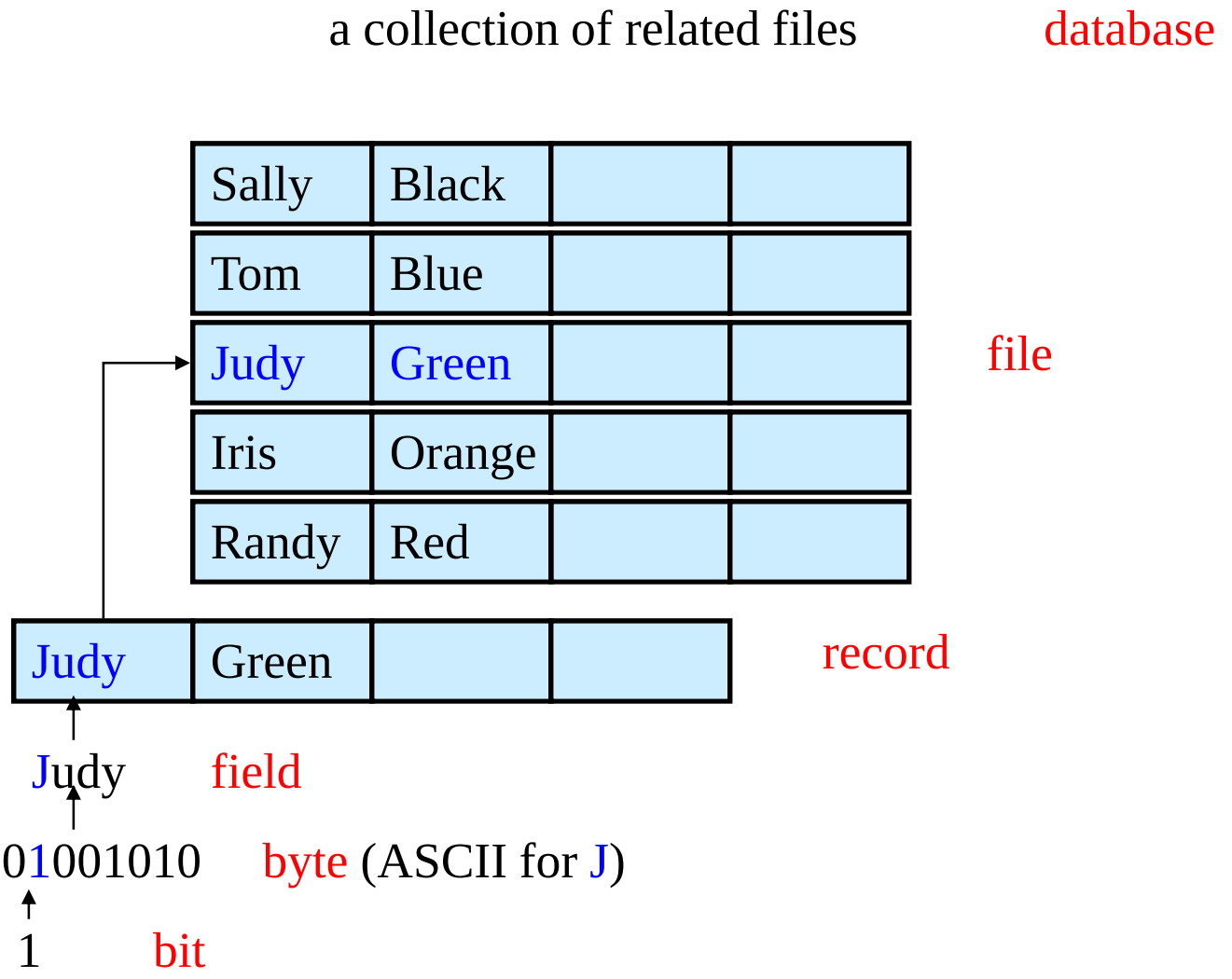


Fig. 18.1 Data hierarchy.

18.3 Files and Streams

- File – a sequential stream of bytes

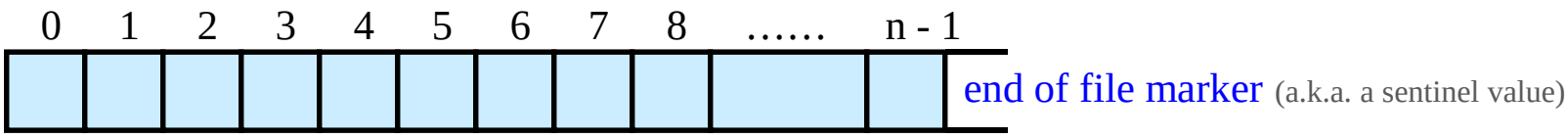


Fig. 18.2 C#'s view of an *n-byte* file.

  - File end determined by:
    - an end-of-file marker
    - file length - number of bytes *n* in the file
      - Recorded in system data structure

- When file opened in C#:
  - Creates an object
  - Associates a stream with that object

18.3 Files and Streams

- When a program starts, 3 stream objects are created:
  - They facilitate communication:

    the program <----> a particular file or device
  - The 3 stream objects are accessed via their properties:
    - Property Console.In: returns standard input stream object
      - Enables a program to input from the keyboard
    - Property Console.Out: returns standard output stream object
      - Enables a program to output to the screen
    - Property Console.Error: returns standard error stream object
      - Enables a program to output error messages to the screen
- We have already used Console.In and Console.Out
  - Console.Read and Console.ReadLine use Console.In implicitly
  - Console.Write and Console.WriteLine use Console.Out implicitly

18.3 Files and Streams

- Namespace **`System.IO`** needed for file processing
  - Includes definitions for stream classes such as:
    - Class StreamReader - for text input from a file
      - inherits from abstract class TextReader
        - Console.In is a property of TextReader
    - Class StreamWriter - for text output to a file
      - inherits from abstract class TextWriter
        - Console.Out is a property of TextWriter
    - Class FileStream - for text input/output from/to a file
      - inherits from abstract class Stream
      - Example:

```
private FileStream output;
output = new FileStream( fileName,
            FileMode.OpenOrCreate, FileAccess.Write );
```

  - Files are opened by creating objects of these stream classes

18.3 Files and Streams

- BinaryFormatter class: converts (serializes or deserializes) original objects from/to the corresponding Stream objects
  - Serialize: Convert an object into a format that can be written to a file without losing any of that object's data

    original object --[BinaryFormatter]--> Stream object

    Note an analogy to the 'ToString' methods that we have used to convert objects to strings. (The 'ToString' conversion was not *for files – for converting in-memory objects to strings only!).*

  - Deserialize: Read a serialized object (serialized into a stream object) from a file and reconstruct the original object from it

    Stream object --[BinaryFormatter]--> original object

18.3 Files and Streams

- Abstract class **`System.IO.Stream`** allows for representation of streams as bytes
  - Bytes (representing streams) are stored in files/memory/buffer

  or
  - Bytes (representing streams) are retrieved from files/memory/buffer
- Concrete classes derived from **`Stream`**:
  - Class **`FileStream`**: read/write to/from sequential-access and random-access files

  [**SKIP**: The following are not used now:]
  - Class **`MemoryStream`**: transfer of data directly to and from memory
    - Much faster than other types of data transfer (e.g., to/from disk)
  - Class **`BufferedStream`**: uses buffering to transfer data to/from a stream
    - Buffering enhances I/O performance
      - Logical output operations (to buffer) add data to the buffer (a region in memory)
      - When buffer full, physical output operation to a device performed

18.3 Files and Streams

- Example sequence of writing an object to a file:

  Step 1: original object --[BinaryFormatter - serialize]--> Stream object

  Step 2: Stream object --[FileStream]--> file

- Example sequence of reading an object from a file:

  Step 1: file --[FileStream]--> Stream object

  Step 2: Stream object --[BinaryFormatter - deserialize]--> original object

*** SKIP ***

18.4 Classes File and Directory

18.5 Creating a Sequential-Access File

- In C#, files have no structure
  - Just a sequence of bytes
    - *No* record structure

- Programmers have to structure files to meet the requirements of applications
  - Define/create your own record structure within a file
    - Using text and special characters

DIGRESSION: Enumeration

- Enumeration declaration:

```
<modifier> enum <enum_name>  : <underlying_type>
                        // modifier: private, public, protected or internal
 {
        <Enumeration list>
 {
```

- Example

```
enum Months                 // default underlying type is int
 {
        JAN, FEB, MAR, APR
 }
```

- By default the first enumerator has the value of 0 and the value of each successive enumerator is increased by 1.
  - Example: In the above case, the value JAN is 0, FEB is 1, …
- Can override default values
  - For example

```
enum Months
 {
        JAN = 10, FEB = 20, MAR = 30, APR = 40
 }
```

- Example: in the above case the value JAN is 0, FEB is 1 and so on
  - Example: Now the value JAN is 10, FEB is 20, …

[cf. Rajesh V. S. - see http://www.c-sharpcorner.com/Language/EnumeratorsInCsharpRVS.asp]

DIGRESSION: Enumeration – cont.

- An explicit cast is needed to convert from underlying enum type to another type, e.g., integer type. (This is why the enum types are type-safe in C#.)
  - Small example

    int x = Months.JAN; is not a valid statement in C#.
    int x = (int) Months.JAN ; is a valid statement.

- Larger example (incl. converting and printing enum values)

```
using System;
enum Months : long          // non-default underlying type – 64-bit size
{
 JAN = 10, FEB = 20
}
class MyClient
{
    public static void Main()
    {
        long x = ( long ) Months.JAN;
        long y = ( long ) Months.FEB;
        Console.WriteLine("JAN value = {0}, FEB value = {1}", x, y);
    }
}
```

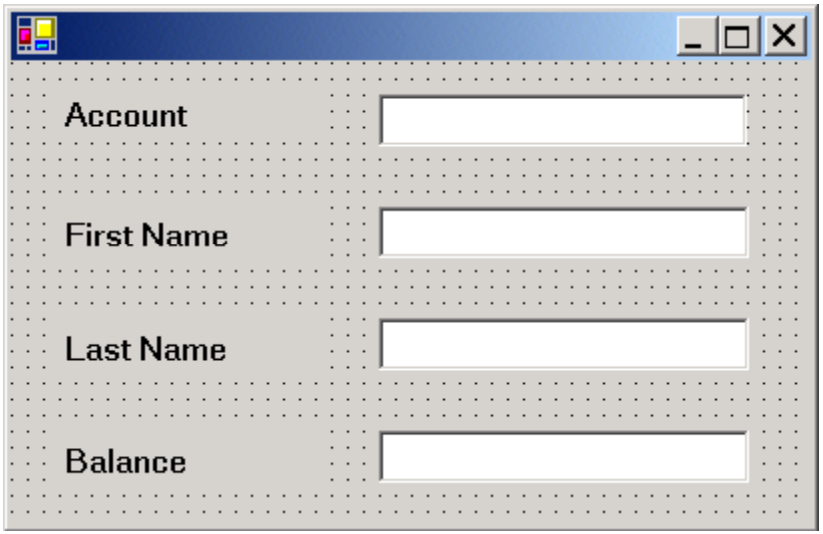
Output:

```
JAN value = 10, FEB value = 20
```

[cf. Rajesh V. S. - see http://www.c-sharpcorner.com/Language/EnumeratorsInCsharpRVS.asp]

18.5 Creating a Sequential-Access File

- Examples in Chapter 18: File processing in a bank-account maintenance application
  - Class BankUIForm - a reusable windows form for examples

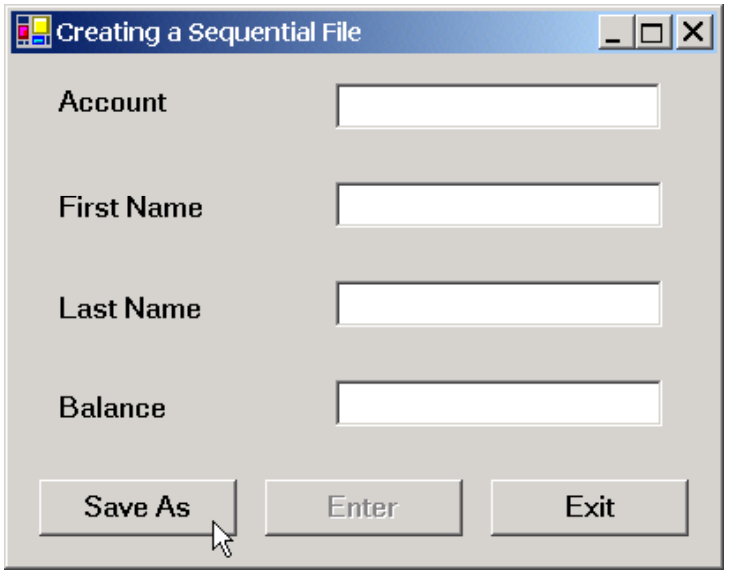


  - Class Record – used for writing/reading records to/from a sequential file (in Fig.17.9, 17.11, 17.12 in the slides below)
  - Both classes compiled into a DLL library named BankLibrary

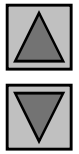
18.5 Creating a Sequential-Access File

- Other classes in examples in this slides Section
  - Derived from BankUIForm:
    - public class CreateFileForm : BankUIForm (Fig 17.9)
    - public class ReadSequentialAccessFileForm : BankUIForm (Fig 17.11)

Figure: GUIs for CreateFileForm and ReadSequentialAccessFileForm add 3 buttons to BankUIForm's GUI (compare this figure with the figure on the preceding slide)



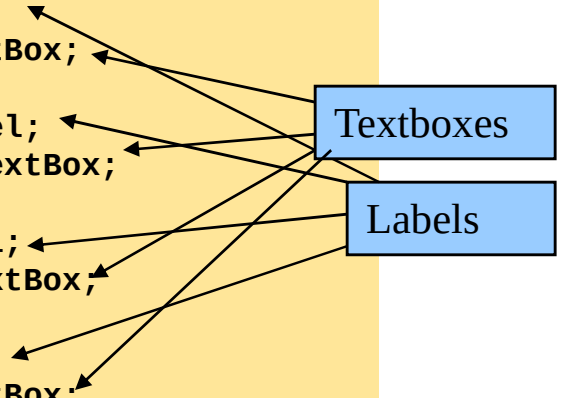
  - Not derived from BankUIForm:
    - public class CreditInquiryForm : Systems.Windows.Forms.Form (Fig. 17.12)



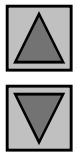
Outline

BankUI.cs

```
1    // Fig 17.7: BankUI.cs
2    // A reusable windows form for the examples in this chapter.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class BankUIForm : System.Windows.Forms.Form
12   {
13      private System.ComponentModel.Container components = null;
14
15      public System.Windows.Forms.Label accountLabel;
16      public System.Windows.Forms.TextBox accountTextBox;
17
18      public System.Windows.Forms.Label firstNameLabel;
19      public System.Windows.Forms.TextBox firstNameTextBox;
20
21      public System.Windows.Forms.Label lastNameLabel;
22      public System.Windows.Forms.TextBox lastNameTextBox;
23
24      public System.Windows.Forms.Label balanceLabel;
25      public System.Windows.Forms.TextBox balanceTextBox;
26
27      // number of TextBoxes on Form'
28      protected int TextBoxCount = 4;    // protected access – between
         // private and public, accessible in this & its derived classes
29
```



Slide modified by L. Lilien



Outline

BankUI.cs

```
30      // enumeration constants specify TextBox indices
31      public enum TextBoxIndices
32      {
33         ACCOUNT,          // - by default, ACCOUNT = 0
34         FIRST,            // - by default, FIRST = 1
35         LAST,             // - by default, LAST = 2
36         BALANCE           // - by default, BALANCE = 3
37                           // Used in Lines 81-88
38      } // end enum  // enumerated type TextBoxIndices used here
39                    // to define field offsets within a record:
40      [STAThread]
41      static void Main()
42      {
43         Application.Run( new BankUIForm() );
44      }
45
46      // Visual Studio .NET generated code comes here
47
48      // clear all TextBoxes
49      public void ClearTextBoxes()
50      {
51         // iterate through every Control on form
52         for ( int i = 0; i < Controls.Count; i++ )
53         {    // Count is 8, since BankUIForm has 8 controls: 4 labels
                // and 4 text boxes  - see lines 15-25 and Slide 15
54            Control myControl = Controls[ i ]; // get control
55
56            // determine whether Control is TextBox
57            if ( myControl is TextBox )
58            {      // skip if not a text box, i.e., if a label
59               // clear Text property (set to empty strng)
60               myControl.Text = "";
61            }
62         }
63
64      } // end method ClearTextBoxes
```

Method to clear textboxes

Slide modified by L. Lilien

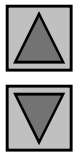
Outline

```
65
66       // set text box values to string array 'values'
67       public void SetTextBoxValues( string[] values )
68       {
69          // determine whether string array has correct leng
70          if ( values.Length != TextBoxCount )
71          {
72             // throw exception if not correct length
                              // - will discuss exceptions next week
73             throw( new ArgumentException( "There must be " +
74                (TextBoxCount + 1) + " strings in the array" ) );
75          }
76
77          // set array values if array has correct length
78          else
79          {
80             // set array values to text box values - using offsets
81             accountTextBox.Text =
82                values[ ( int )TextBoxIndices.ACCOUNT ]; // cast required
83             firstNameTextBox.Text =                     // see Slide 14
84                values[ ( int )TextBoxIndices.FIRST ];
85             lastNameTextBox.Text =
86                values[ ( int )TextBoxIndices.LAST ];
87             balanceTextBox.Text =
88                values[ ( int )TextBoxIndices.BALANCE ];
89          }
90
91       } // end method SetTextBoxValues
92
93       // set string array 'values' to values entered (as text) in text
         // boxes
94       public string[] GetTextBoxValues()
95       {
96          string[] values = new string[ TextBoxCount ];
97
```

Method to set values of textboxes

I.cs

Method to get the values of textboxes

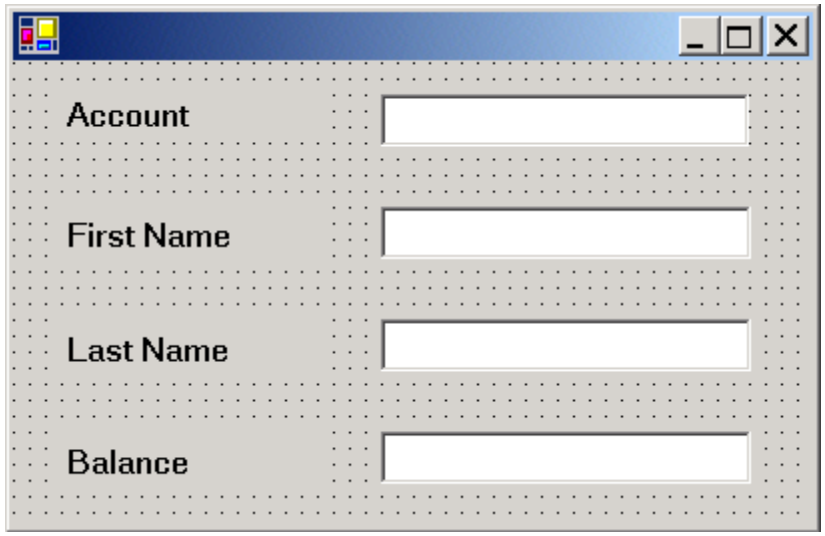


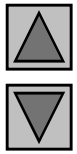
Outline

BankUI.cs

```
98          // copy text box fields to string array - using offsets
99          values[ ( int )TextBoxIndices.ACCOUNT ] =
100            accountTextBox.Text;
101         values[ ( int )TextBoxIndices.FIRST ] =
102            firstNameTextBox.Text;
103         values[ ( int )TextBoxIndices.LAST ] =
104            lastNameTextBox.Text;
105         values[ ( int )TextBoxIndices.BALANCE ] =
106            balanceTextBox.Text;
107
108         return values;
109
110      } // end method GetTextBoxValues
111
112   } // end class BankUIForm
```

Program Output





Outline

Record.cs

```
1    // Fig. 17.8: Record.cs
2    // Serializable class that represents a data record.
3
4    using System;
5
6  [Serializable]
7    public class Record
8    {
9       private int account;
10      private string firstName;
11      private string lastName;
12      private double balance;
13
14      // default constructor sets members to default values
15      public Record() : this( 0, "", "", 0.0 )
16      {
17      }
18
19      // overloaded constructor sets members to parameter values
20      public Record( int accountValue, string firstNameValue,
21         string lastNameValue, double balanceValue )
22      {
23         Account = accountValue;
24         FirstName = firstNameValue;
25         LastName = lastNameValue;
26         Balance = balanceValue;
27
28      } // end constructor
29
```

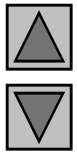
Required: Tells compiler objects of class **Record** can be represented as sets of bytes (can read/write bytes from/to streams)

Data to go into record

Sets members to 0

Set members to parameters

Slide modified by L. Lilien

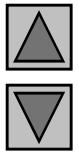


Outline

Record.cs

```
30      // property Account
31      public int Account
32      {
33         get
34         {
35            return account;
36         }
37
38         set
39         {
40            account = value;
41         }
42
43      } // end property Account
44
45      // property FirstName
46      public string FirstName
47      {
48         get
49         {
50            return firstName;
51         }
52
53         set
54         {
55            firstName = value;
56         }
57
58      } // end property FirstName
59
```

Accessor methods



Outline

Record.cs

```
60      // property LastName
61      public string LastName
62      {
63         get
64         {
65            return lastName;
66         }
67
68         set
69         {
70            lastName = value;
71         }
72
73      } // end property LastName
74
75      // property Balance
76      public double Balance
77      {
78         get
79         {
80            return balance;
81         }
82
83         set
84         {
85            balance = value;
86         }
87
88      } // end property Balance
89
90   } // end class Record
```

Accessor methods

GUI for the Next Program (Fig. 17.9 - CreateSequentialAccessFile.cs):

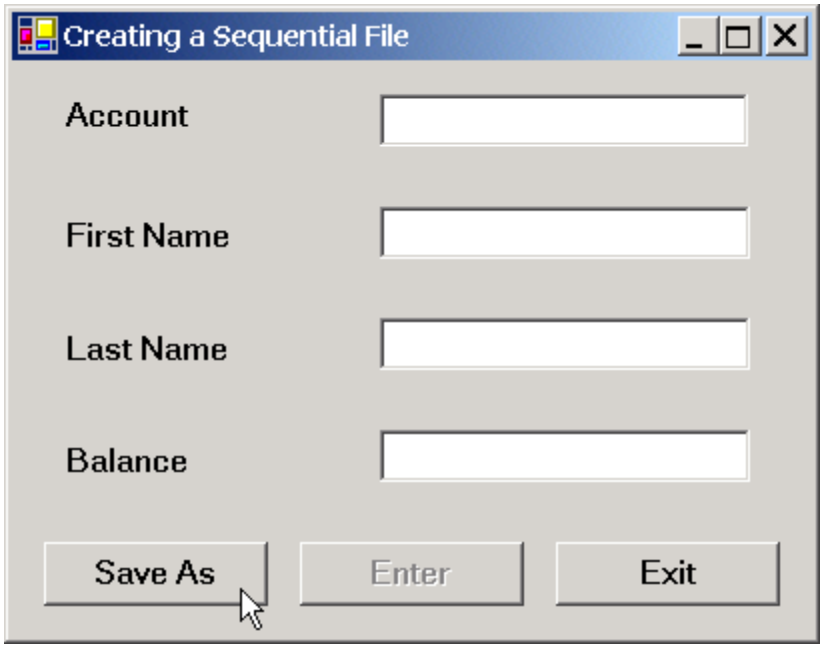
Creating a sequential-access file = **Writing data into** a sequential-access file **– Example**

Outline

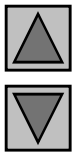
CreateSequentialAccessFile.cs
Program Output

User clicks the "Save As" button in the "Creating a Sequential File" window to start file creation process.

File selection dialog window (next slide) appears.



Slide modified by L. Lilien

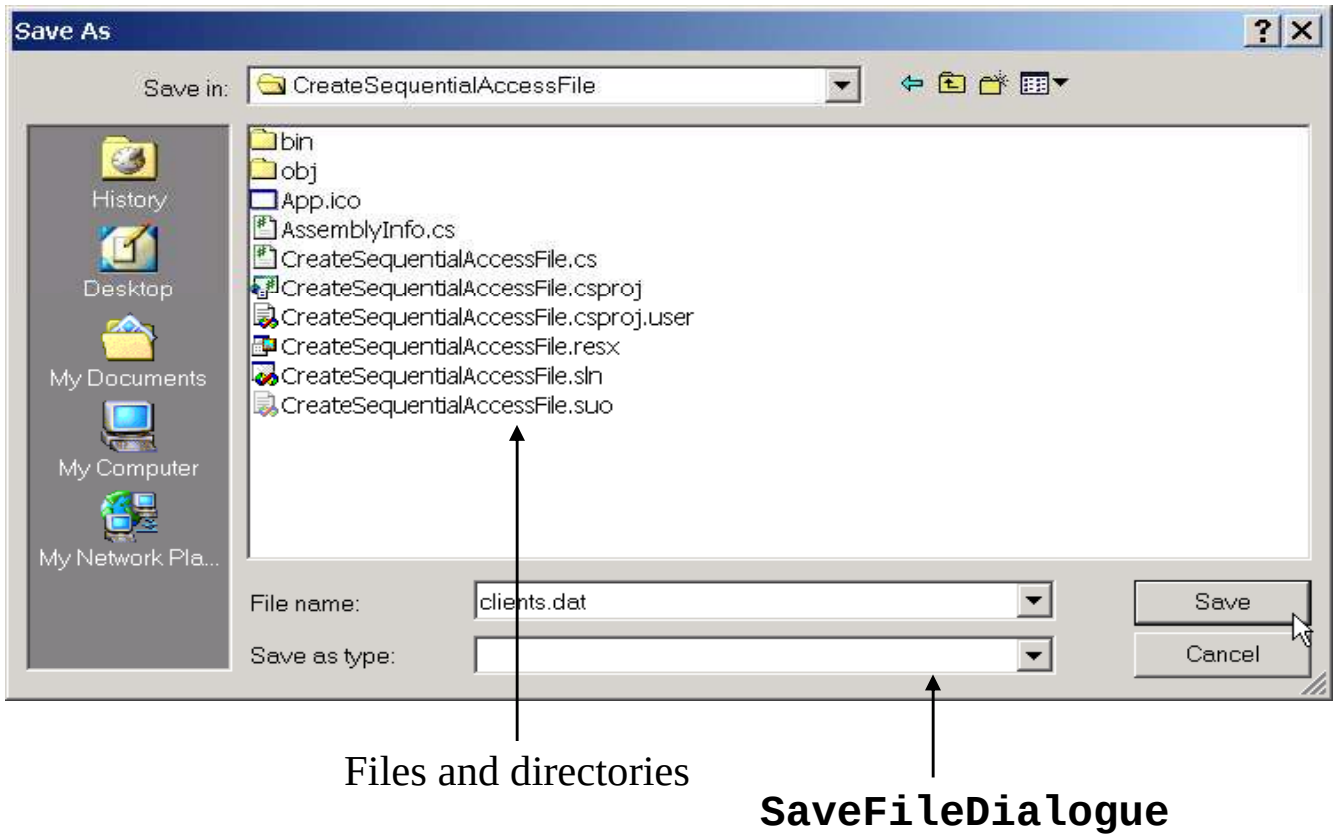


Outline

CreateSequentialAccessFile.cs

Program Output

User creates a file by selecting its drive & directory (by pointing & clicking), and by typing its filename ‘clients.dat’ (into the ‘File name:’ text box) in the displayed file selection dialog window named “Save As”.



Slide modified by L. Lilien

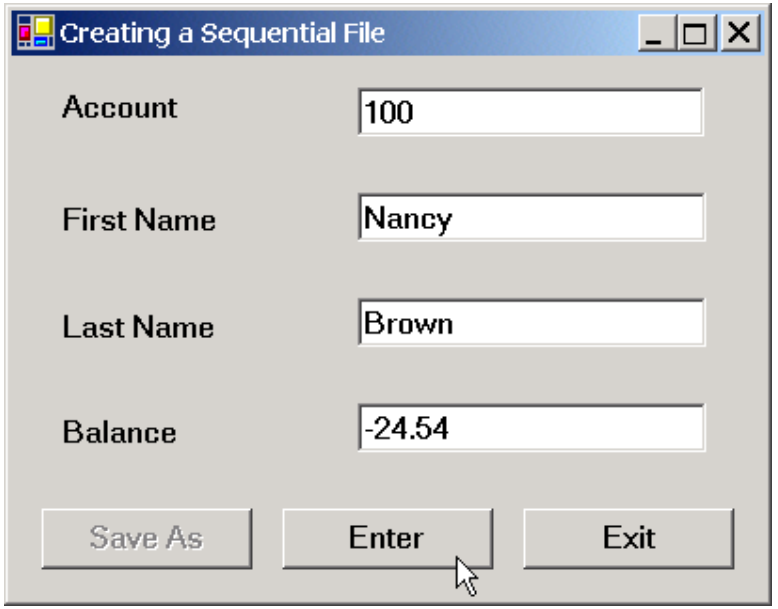
Outline

After the file location and name are entered, data (records) are entered as follows:

User enters data into 4 textboxes, then clicks on 'Enter'.

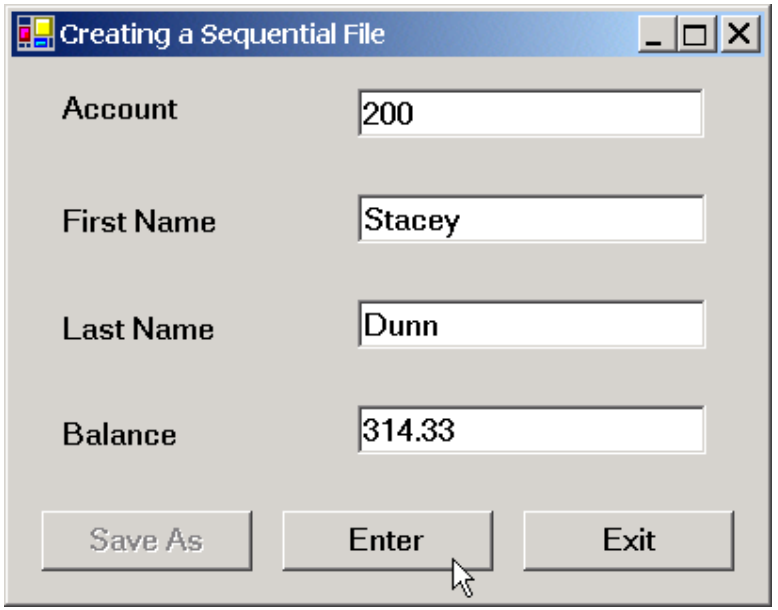
(For simplicity, user enters data in the 'Account' number order; 'Account' is the record key)

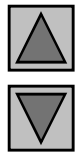
Program stores data in 4 fields of Record 1 in a file.



User enters data into 4 textboxes, then clicks on 'Enter'.

Program stores data in 4 fields of Record 2 in a file.

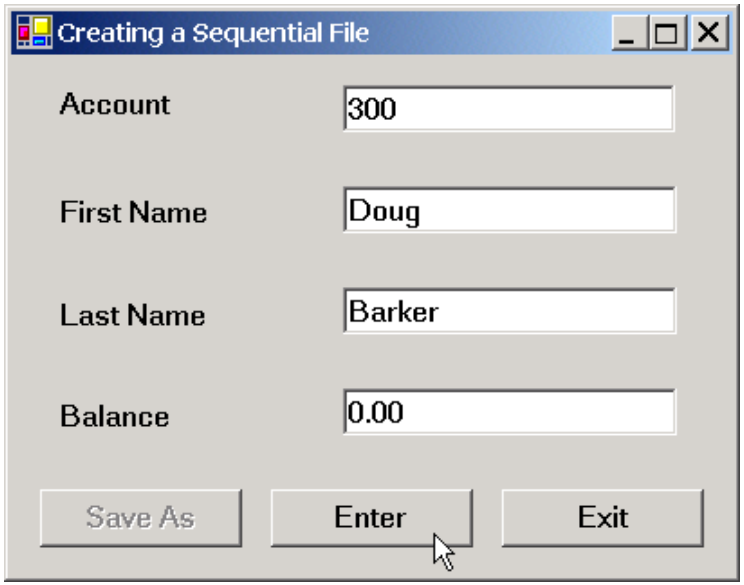




Outline

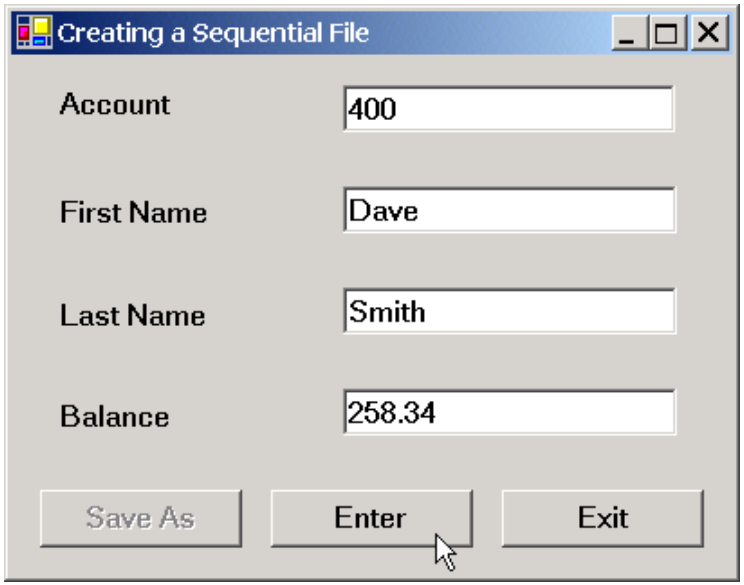
User enters data into 4 textboxes, then clicks on 'Enter'.

Program stores data in 4 fields of Record 3 in a file.

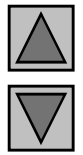


User enters data into 4 textboxes, then clicks on 'Enter'.

Program stores data in 4 fields of Record 4 in a file.



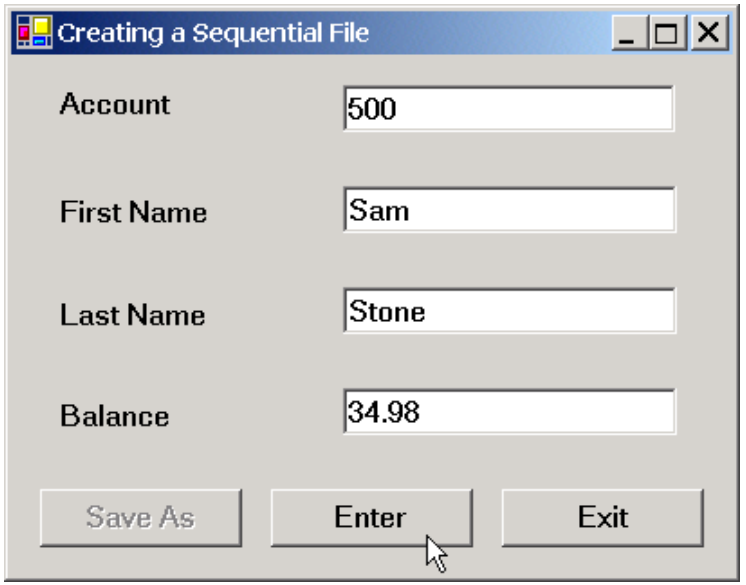
Slide added by L. Lilien



Outline

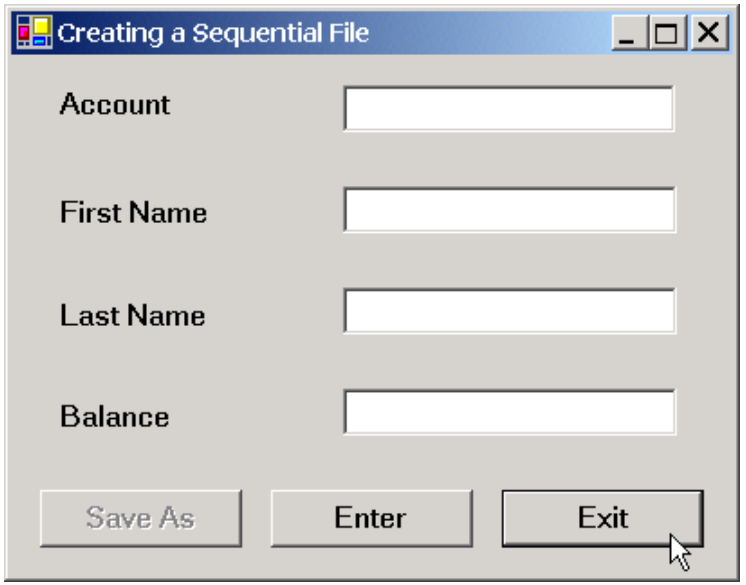
User enters data into 4 textboxes, then clicks on 'Enter'.

Program stores data in 4 fields of Record 5 in a file.



User clicks on 'Exit' to terminate program execution.

Program terminates, but all five records remain stored safely.



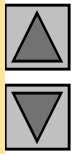
Slide added by L. Lilien

Outline

eateSequential
cessFile.cs

```
1   // Fig 17.9: CreateSequentialAccessFile.cs
2   // Creating a sequential-access file.
3
4   // C# namespaces
5   using System;
6   using System.Drawing;
7   using System.Collections;
8   using System.ComponentModel;
9   using System.Windows.Forms;
10  using System.Data;
11  using System.IO;
12  using System.Runtime.Serialization.Formatters.Binary;
13  using System.Runtime.Serialization;
14
15  // Deitel namespace
16  using BankLibrary;
17
18  public class CreateFileForm : BankUIForm
19  {                    // 3 additional buttons added to the inherited window
20     private System.Windows.Forms.Button saveButton;  // 'Save As' button
21     private System.Windows.Forms.Button enterButton; // 'Enter' button
22     private System.Windows.Forms.Button exitButton;  // 'Exit' button
23       // The 3 buttons appear in the new "Creating a Sequential File" window
24     private System.ComponentModel.Container components = null;
25
26     // serializes Record in binary format
27     private BinaryFormatter formatter = new BinaryFormatter();
28             // BinaryFormatter - see Slide 8
29     // stream through which serialized data is written to file
30     private FileStream output; // FileStream - see Slide 7
31
```

Slide modified by L. Lilien



Outline

CreateSequentialAccessFile.cs

```
32      [STAThread]
33      static void Main()
34      {
35         Application.Run( new CreateFileForm() );
36      }
37
38      // Visual Studio .NET generated code
39       // 'saveButton_Click' method is for 'Save As' button in the 'Creating
         // a Sequential File' window (not 'Save' button in 'Save As' window)
40      // invoked when user clicks Save button
41      private void saveButton_Click(
42         object sender, System.EventArgs e )
43      {
44         // create dialog box enabling user to save file
45         SaveFileDialog fileChooser = new SaveFileDialog();
               // Object fileChooser of SaveFileDialog class used to
               // select files - see Slide 25
46         DialogResult result = fileChooser.ShowDialog();
               // 'ShowDialog()' displays fileChooser object
              // and returns an integer, which is stored in 'result'.
47         string fileName; // declare variable for holding the
                            // name of file to save data
48
49         // create file for user (or open if it already exists)- unless
           // user cancels or provides wrong file name
50         fileChooser.CheckFileExists = false;
51
52         // exit event handler if user clicked "Cancel"
53         if ( result == DialogResult.Cancel )
54            return;
55                // we're here only if user did not click "Cancel"
56         // get specified file name
               // using property Filename of fileChooser
57         fileName = fileChooser.FileName;
58
59         // show error if user specified invalid file
60         if ( fileName == "" || fileName == null )
61            MessageBox.Show( "Invalid File Name", "Error",
62               MessageBoxButtons.OK, MessageBoxIcon.Error );
```

Instantiate SaveFileDialog object

Show SaveFileDialog

Test if user canceled save

Get file name to save to

Slide modified by L. Lilien

line

```
63          else
64          {
65             // save file via FileStream if user specified valid file
66             try
67             {
68                // open file with write access
                  // opens file by instantiating a FileStream object – see Slide 7
69                output = new FileStream( fileName,
70                   FileMode.OpenOrCreate, FileAccess.Write );
                      // - FileMode.OpenOrCreate = open if exists, create otherwise
                      // - FileAccess.Write = allow only write operations on 'output';
                      // other possible values: FileAccess.Read, FileAccess.ReadWrite
71
72                // disable Save button and enable Enter button
73                saveButton.Enabled = false;
74                enterButton.Enabled = true;
75             }
76
77             // handle exception if file fileName (l. 69) does not exist
78             catch ( FileNotFoundException )
79            {
80                // notify user if file does not exist
81                MessageBox.Show( "File Does Not Exist", "Error",
82                   MessageBoxButtons.OK, MessageBoxIcon.Error );
83             }
84          }
85       } // end method saveButton_Click
86
87       // invoke when user clicks Enter button
88       private void enterButton_Click(
89          object sender, System.EventArgs e )
90       {
91          // store TextBox values string array
92          string[] values = GetTextBoxValues(); // defined by us - see Slides 19-20
93
94          // Store values from text boxes into file (organized
            // into records)
95          Record record = new Record();  //instantiate a new record
96
```

Instantiate output stream with write permission

Method to save data when user clicks enter

Slide modified by L. Lilien

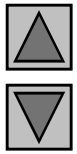
Sequential File.cs

```
97          // a lot to do if TextBox account field is not empty
98          if ( values[ ( int ) TextBoxIndices.ACCOUNT ] != "" )  // explicit cast
99          {                                                      // for enum
100            // store TextBox values in record and serialize record
101            try
102            {
103               // get account number value from TextBox
104               int accountNumber = Int32.Parse(
105                  values[ ( int ) TextBoxIndices.ACCOUNT ] );
106                     // convert the string values […] into an int
107               // determine whether accountNumber is valid
108               if ( accountNumber > 0 )
109               {
110                  // store TextBox fields in the object record (of type
                     // Record)
111                  record.Account = accountNumber;
112                  record.FirstName =
113                     values[ ( int ) TextBoxIndices.FIRST ];
114                  record.LastName =
115                     values[ ( int ) TextBoxIndices.LAST ];
116                  record.Balance = Double.Parse(
117                     values[ ( int ) TextBoxIndices.BALANCE ] );
118
119                  // serialize object: write Record to the object
                     // output (of type FileStream) – see l.30
                         // by using method serialize for obj. formatter
                         // of type BinaryFormatter – see l.27
120                  formatter.Serialize( output, record );
121               }
122               else
123               {
124                  // notify user if invalid account number
125                  MessageBox.Show( "Invalid Account Number", "Error",
126                     MessageBoxButtons.OK, MessageBoxIcon.Error );
127               }
128            } // end of try
129
```

Store TextBox fields in record

Serialize `record` into `output` (of type FileStream)

Slide modified by L. Lilien



Outline

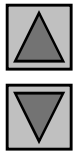
CreateSequentialAccessFile.cs

```
130          // notify user if error occurs in serialization
131          catch( SerializationException )
132          {
133             MessageBox.Show( "Error Writing to File", "Error",
134                MessageBoxButtons.OK, MessageBoxIcon.Error );
135          }
136
137          // notify user if error occurs regarding parameter format
138          catch( FormatException )
139          {
140             MessageBox.Show( "Invalid Format", "Error",
141                MessageBoxButtons.OK, MessageBoxIcon.Error );
142          }
143       }
144
145       ClearTextBoxes(); // clear TextBox values
                           // Method def. in class BankUIForm – l.49
146
147    } // end method enterButton_Click
148
149    // invoked when user clicks Exit button
150    private void exitButton_Click(
151       object sender, System.EventArgs e )
152    {
153       // determine whether file exists
154       if ( output != null )
155       {
156          // close file
157          try
158          {
159             output.Close();
160          }
161
```

Catch block if user input invalid data

Close output stream (of type FileStream – see l. 30)

Slide modified by L. Lilien



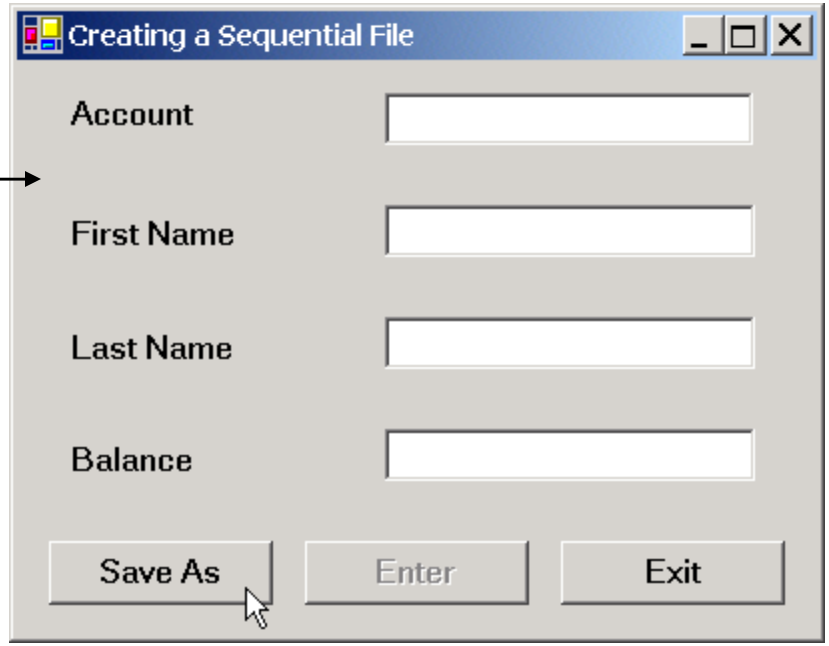
Outline

CreateSequentialAccessFile.cs

```
162           // notify user if error closing file
163           catch( IOException )
164           {
165              MessageBox.Show( "Cannot close file", "Error",
166                 MessageBoxButtons.OK, MessageBoxIcon.Error );
167           }
168        }
169
170        Application.Exit();
171
172     } // end method exitButton_Click
173
174  } // end class CreateFileForm
```

Exit program

BankUI graphical user interface



Program Output

18.5 Creating a Sequential-Access File

Account Number	First Name	Last Name	Balance
100	Nancy	Brown	-25.54
200	Stacey	Dunn	314.33
300	Doug	Barker	0.00
400	Dave	Smith	258.34
500	Sam	Stone	34.98

Fig. 17.10 Sample data for the program of Fig. 17.9.

AGAIN:

GUI for the Preceding Program (Fig. 17.9 - CreateSequentialAccessFile.cs):

Creating a sequential-access file = **Writing data into** a sequential-access file **– Example**

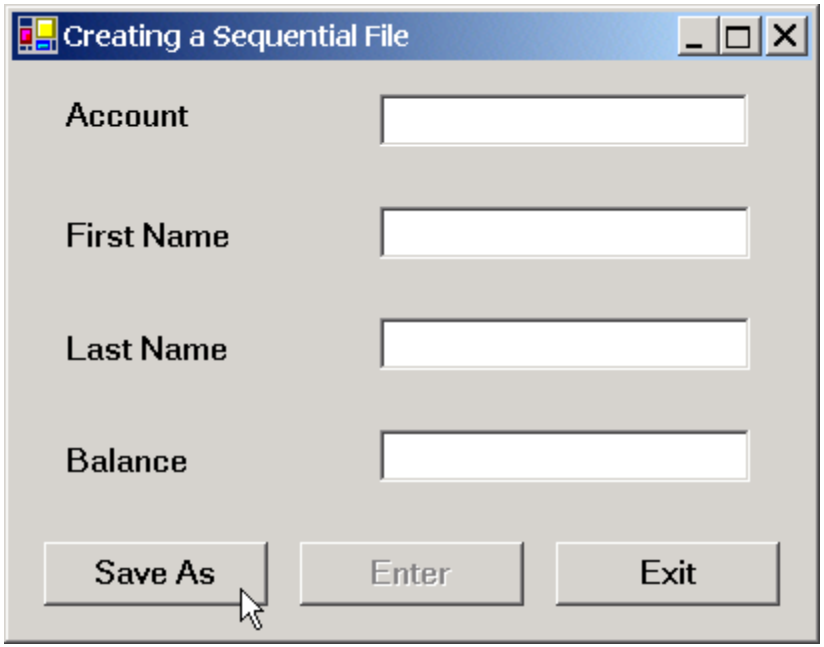
Outline

CreateSequentialAccessFile.cs

Program Output

User clicks the "Save As" button in the "Creating a Sequential File" window to start file creation process.

File selection dialog window (next slide) appears.



Slide modified by L. Lilien

Outline

CreateSequentialAccessFile.cs

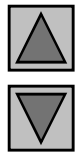
Program Output

SaveFileDialogue

Files and directories

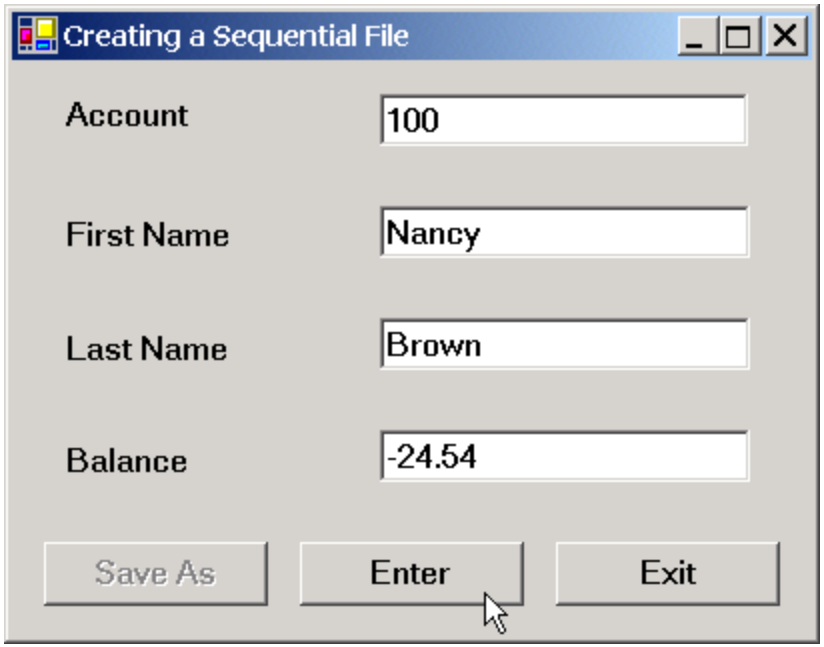
The user selects by clicking the appropriate drive, directory, and types filename ('clients.dat').
Then she clicks Save (or Cancel).

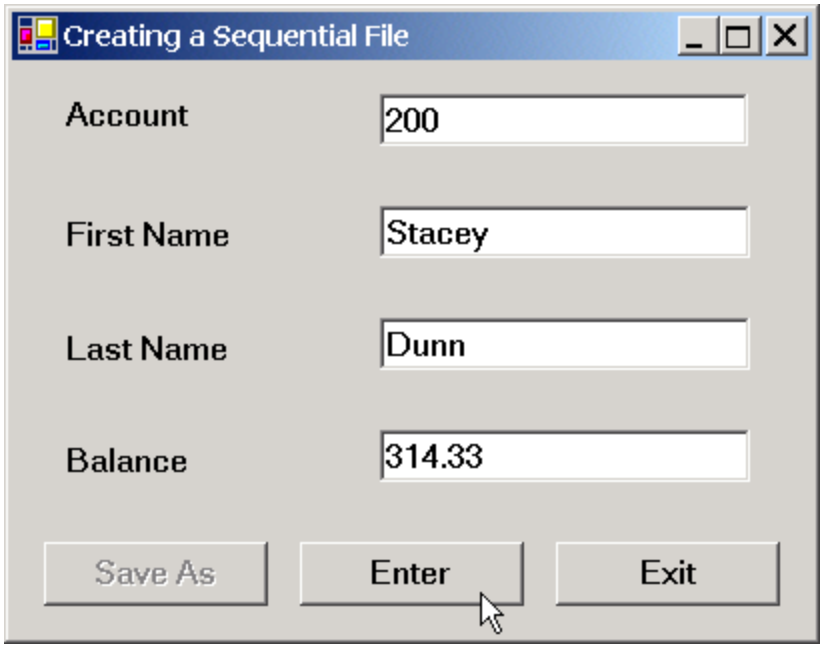
Slide modified by L. Lilien

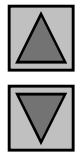


Outline

`CreateSequentialAccessFile.cs`
Program Output

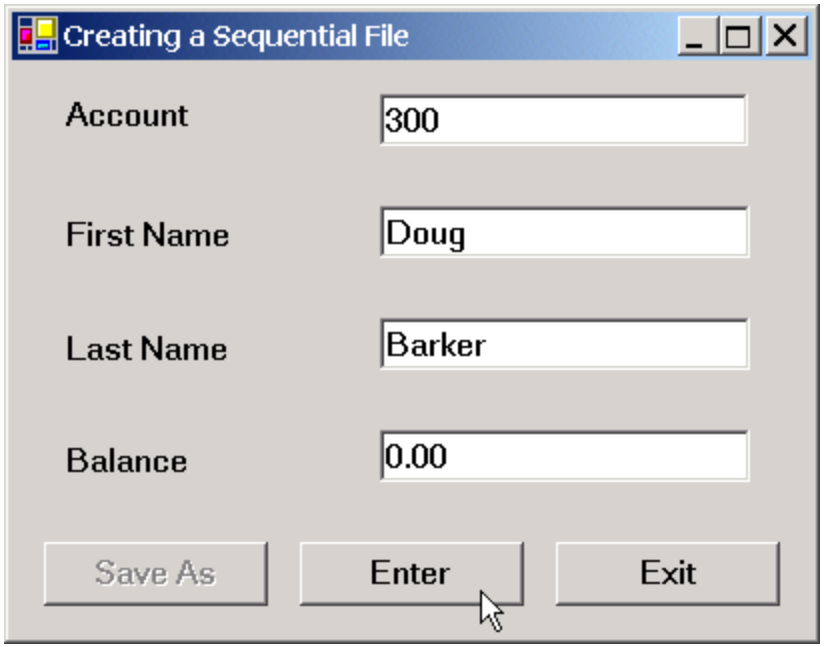


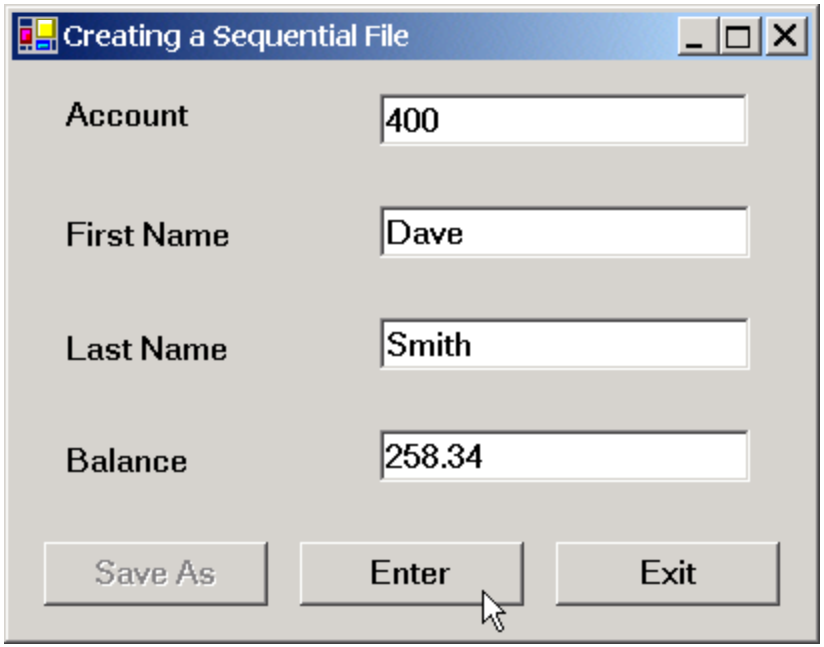


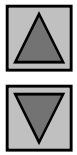


Outline

`CreateSequentialAccessFile.cs`
Program Output

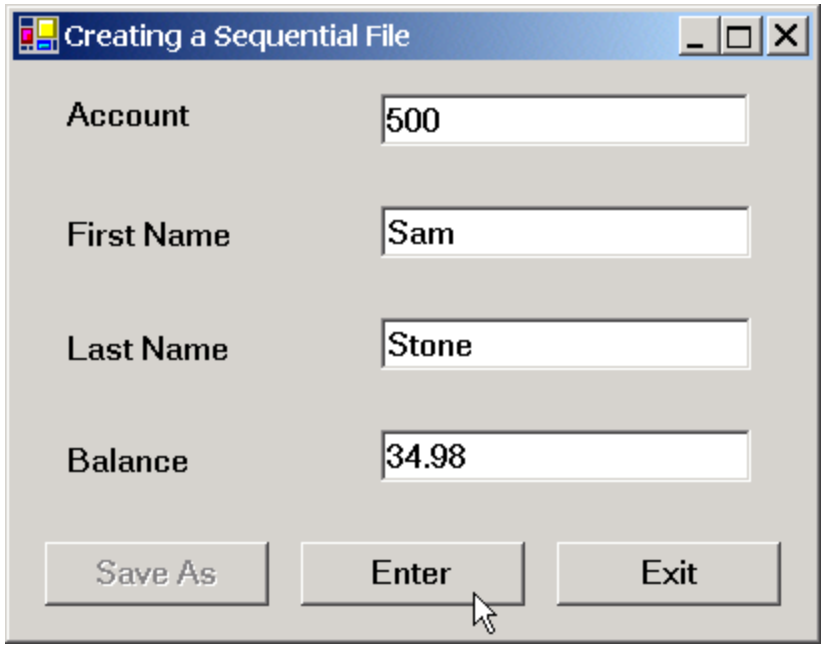


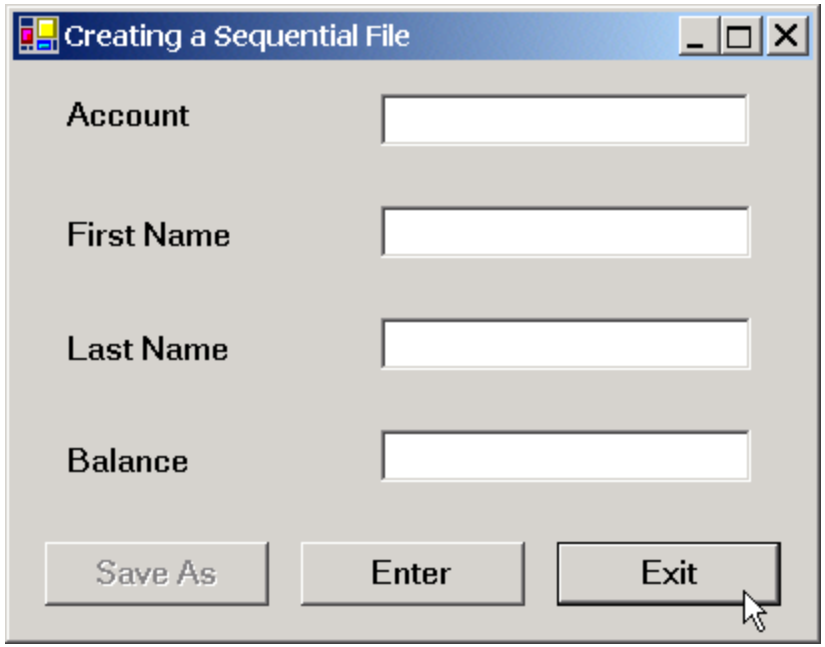




Outline

`CreateSequentialAccessFile.cs`
Program Output





18.6 Reading Data from a Sequential-Access File

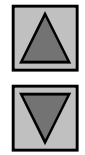
- Read data sequentially from a file
  - Program starts at the beginning of a file & read data consecutively until the desired data is found
    - Sometimes necessary to do this several times during execution of a program
  - File-position pointer:
    - When a file (a FileStream object) is opened, its file-position pointer is set to '0' (i.e., points to the first byte – the byte at offset '0' – see Slide 5)
    - Always points to next byte to be read from or written to file
    - Can be repositioned to any point only in a random access file (not covered in CS 1120)

18.6 Reading Data from a Sequential-Access File

- One program can create (or write) a file, and terminate.
- Another program can read the file at any time
  - In a minute / day / week / … / year / …
  - As long as the file is not deleted
- Example:

  public class ReadSequentialAccessFileForm : BankUIForm
  - Reads the same file that CreateSequentialAccessFile (of Fig. 17.9) created (wrote)

GUI for the Next Program (Fig. 17.11 - ReadSequentialAccessFile.cs): **Reading from** a sequential-access file - **Example**



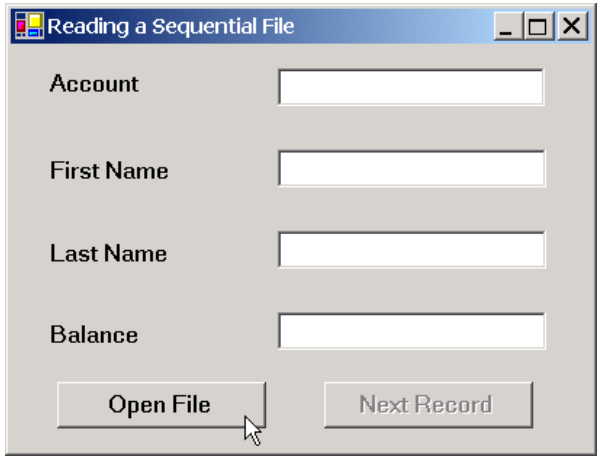
Outline

When the program starts, it displays the "Reading a Sequential File" form.

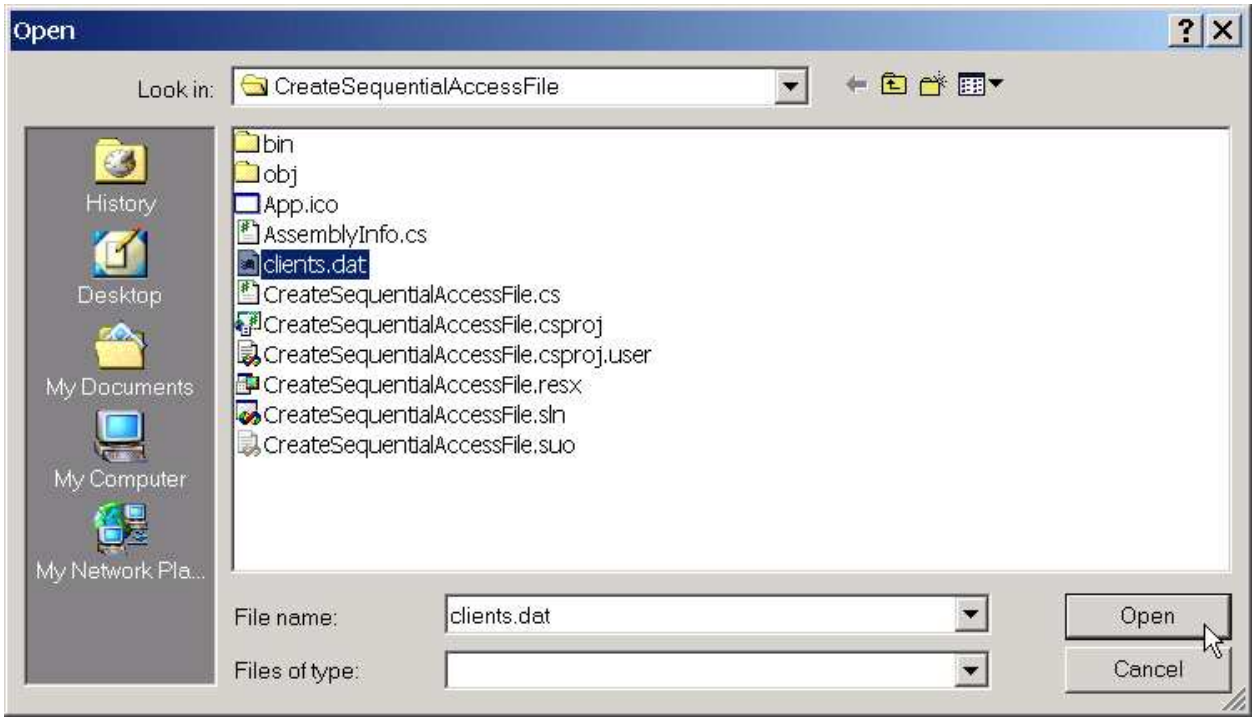
User clicks the "Open File" button in the "Reading a Sequential File" window to start file opening process.

(Notice that "Next Record" is dimmed)

File selection dialog window appears (below).



User opens file by selecting its drive, directory and name - by pointing & clicking in the displayed file selection dialog window named "Open".



Slide added by L. Lilien

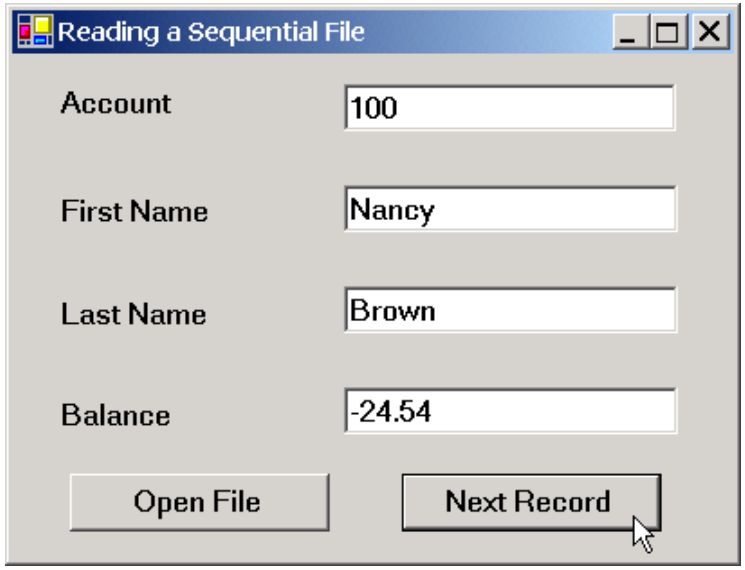
Outline

After the file is opened, data (records) from the file are read as follows:

User clicks on 'Next Record'.

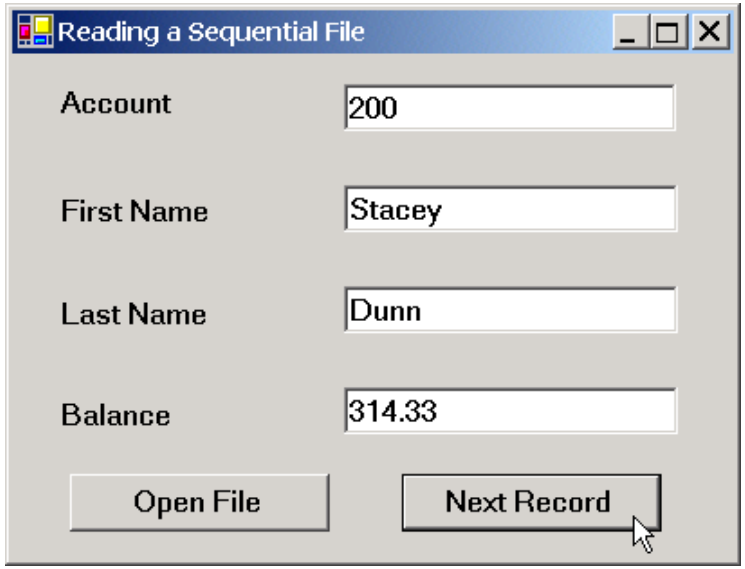
Program displays Record 1, that is, 4 fields of Record 1 from the file.

(As you remember, 'Account' is the record key and data were stored in the file in the 'Account' number order. Thus, the first record displayed is the one with the lowest key: 100.)

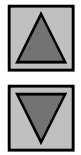


User clicks on 'Next Record'.

Program displays Record 2 (key: 200), that is, 4 fields of Record 2 from the file.



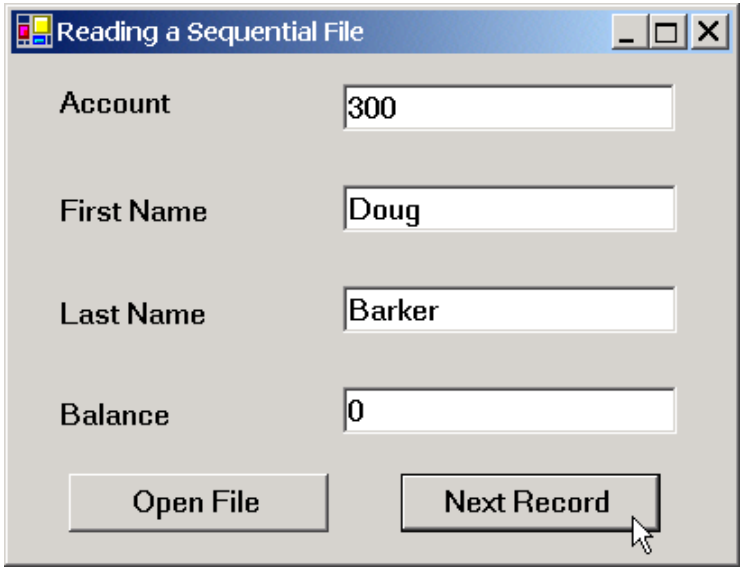
Slide added by L. Lilien



Outline

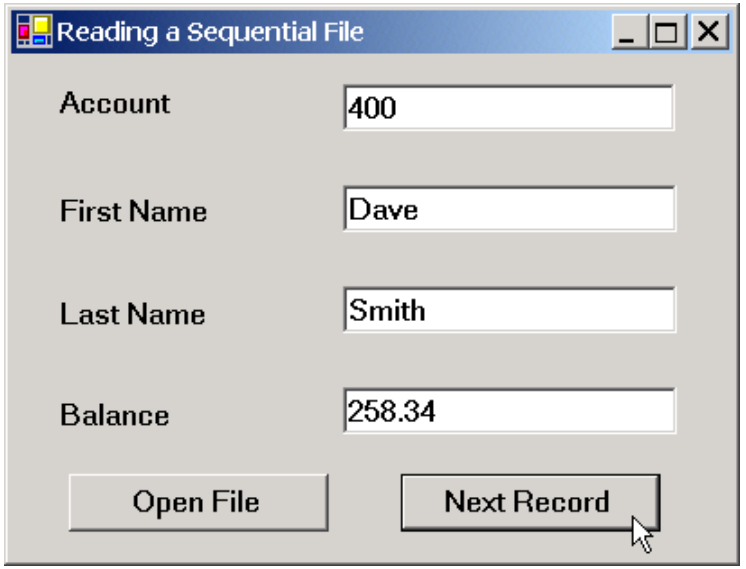
User clicks on ‘Next Record’.

Program displays Record 3 (key: 300), that is, 4 fields of Record 3 from the file.

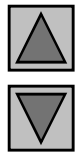


User clicks on ‘Next Record’.

Program displays Record 4 (key: 400), that is, 4 fields of Record 4 from the file.



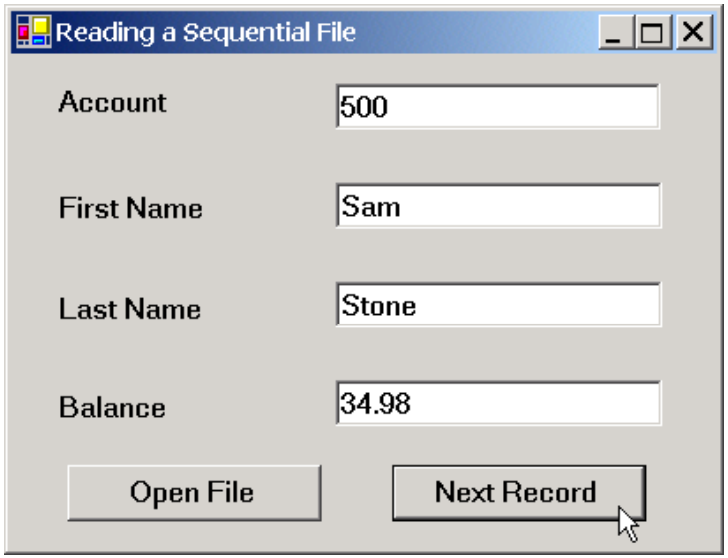
Slide added by L. Lilien



Outline

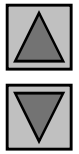
User clicks on ‘Next Record’.

Program displays Record 5 (key: 500), that is, 4 fields of Record 5 from the file.



User clicks on ‘Next Record’.

Program ‘No more records in file’ since end of file has been reached.



Outline

ReadSequentialAccessFile.cs

```
1   // Fig. 17.11: ReadSequentialAccessFile.cs
2   // Reading a sequential-access file.
3
4   // C# namespaces
5   using System;
6   using System.Drawing;
7   using System.Collections;
8   using System.ComponentModel;
9   using System.Windows.Forms;
10  using System.Data;
11  using System.IO;
12  using System.Runtime.Serialization.Formatters.Binary;
13  using System.Runtime.Serialization;
14
15  // Deitel namespaces
16  using BankLibrary;
17
18  public class ReadSequentialAccessFileForm : BankUIForm
19  {                  // 2 additional buttons added to the inherited window
20     System.Windows.Forms.Button openButton;  // “Open File” button
21     System.Windows.Forms.Button nextButton;  // “Next Record” button
22   // The 2 buttons appear in the new “Reading a Sequential File” window
23     private System.ComponentModel.Container components = null;
24
25     // stream through which serializable data are read from file
26     private FileStream input;
27
28     // object for deserializing Record in binary format
29     private BinaryFormatter reader = new BinaryFormatter();
30
31     [STAThread]
32     static void Main()
33     {
34        Application.Run( new ReadSequentialAccessFileForm() );
35     }
```

Slide modified by L. Lilien

Outline

SequentialAc
File.cs

```
36
37      // Visual Studio .NET generated code
38   // The 'openButton_Click' method is for the "Open File" button in "Reading
     // a Sequential File" window (not the "Open" button in the "Open" window)
39      // invoked when user clicks Open button
40      private void openButton_Click(
41         object sender, System.EventArgs e )
42      {
43         // create dialog box enabling user to open file
44         OpenFileDialog fileChooser = new OpenFileDialog();
            // next line displays the "Open"  window - see Slide 43 - bottom
45         DialogResult result = fileChooser.ShowDialog();
46         string fileName; // variable to store name of file
47                          // containing data
48         // exit event handler if user clicked Cancel
49         if ( result == DialogResult.Cancel )
50            return;
51              // we are here if user selected a file
52         // get specified file name (from the FileName property
                                       // of the fileChooser object)
53         fileName = fileChooser.FileName;
54         ClearTextBoxes();     // method of the class BankUIForm - see
55                               // Slide 18
56         // show error if user specified invalid file
57         if ( fileName == "" || fileName == null )
58            MessageBox.Show( "Invalid File Name", "Error",
59               MessageBoxButtons.OK, MessageBoxIcon.Error );
60         else
61         {
62            // create FileStream to obtain read access to file
63            input = new FileStream( fileName, FileMode.Open,
64               FileAccess.Read );
65
66            // enable 'Next Record' button - now it makes sense since the
              // sequential file to be read is already opened - it "un-dism" the button
67            nextButton.Enabled = true;
68         }
69
70      } // end method openButton_Click
```

Instantiate OpenFilelDialog

Display OpenFileDialog

Create `FileStream` object named `input` for input with read only permission

Slide modified by L. Lilien

Outline

eadSequentialAc
essFile.cs

```
71
72       // invoked when user clicks Next button
73       private void nextButton_Click(
74          object sender, System.EventArgs e )
75       {
76          // deserialize Record and store data in TextBoxes
77          try       // enable exception handling
78          {
79             // get next Record from the file
80             Record record =
81                ( Record )reader.Deserialize( input ); // Deserialize
82       // returns object of type 'Object'; cast into 'Record' is necesary
83             // store Record values in temporary string array
84             string[] values = new string[] {
                     // below, ToString converts object to string
85                record.Account.ToString(),    // Account is an int -
                                                 // see l. 104, Slide 32
86                record.FirstName.ToString(),
87                record.LastName.ToString(),
88                record.Balance.ToString() };  // Balance is double -
89                                               // - see l. 116, Slide 32
90             // display string array values within TextBoxes
91             SetTextBoxValues( values );  // we defined this method of the
92          } // end of try block           // class BankUIForm - see Slide 19
93
94          // handle exception when no more Records in file
95          catch( SerializationException )
96          {
97             // close FileStream if no Records in file
98             input.Close();
99
100            // enable 'Open File' button
101            openButton.Enabled = true;
102
103            // disable 'Next Record' button - no more records in file
104            nextButton.Enabled = false;
105
```

Method to view next record from file

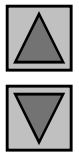
Deserialize and cast next record

Convert record fields into strings

Exception thrown when there are no more records

Close FileStream

Slide modified by L. Lilien



Outline

ReadSequentialAccessFile.cs

```
106           ClearTextBoxes();  // method of the class BankUIForm
107
108           // notify user if no Records in file
109           MessageBox.Show( "No more records in file", "",
110              MessageBoxButtons.OK, MessageBoxIcon.Information );
111        } // catch
112
113     } // end method nextButton_Click
114
115  } // end class ReadSequentialAccessFileForm
```

Tell user there are no more records

Repeating:

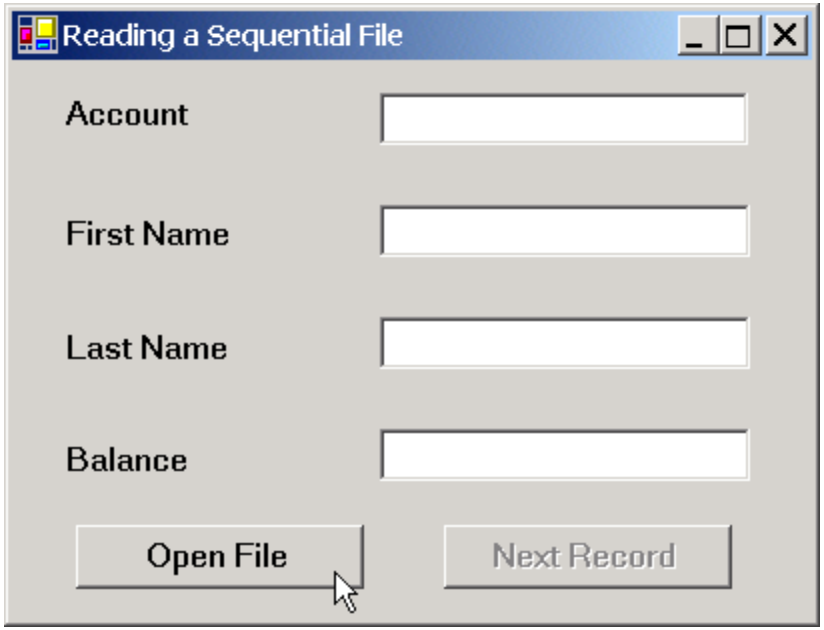
When the program starts, it displays the "Reading a Sequential File" form.

User clicks the "Open File" button to open a file with records.

(Notice that "Next Record" is dimmed)

File selection dialog window appears (below).

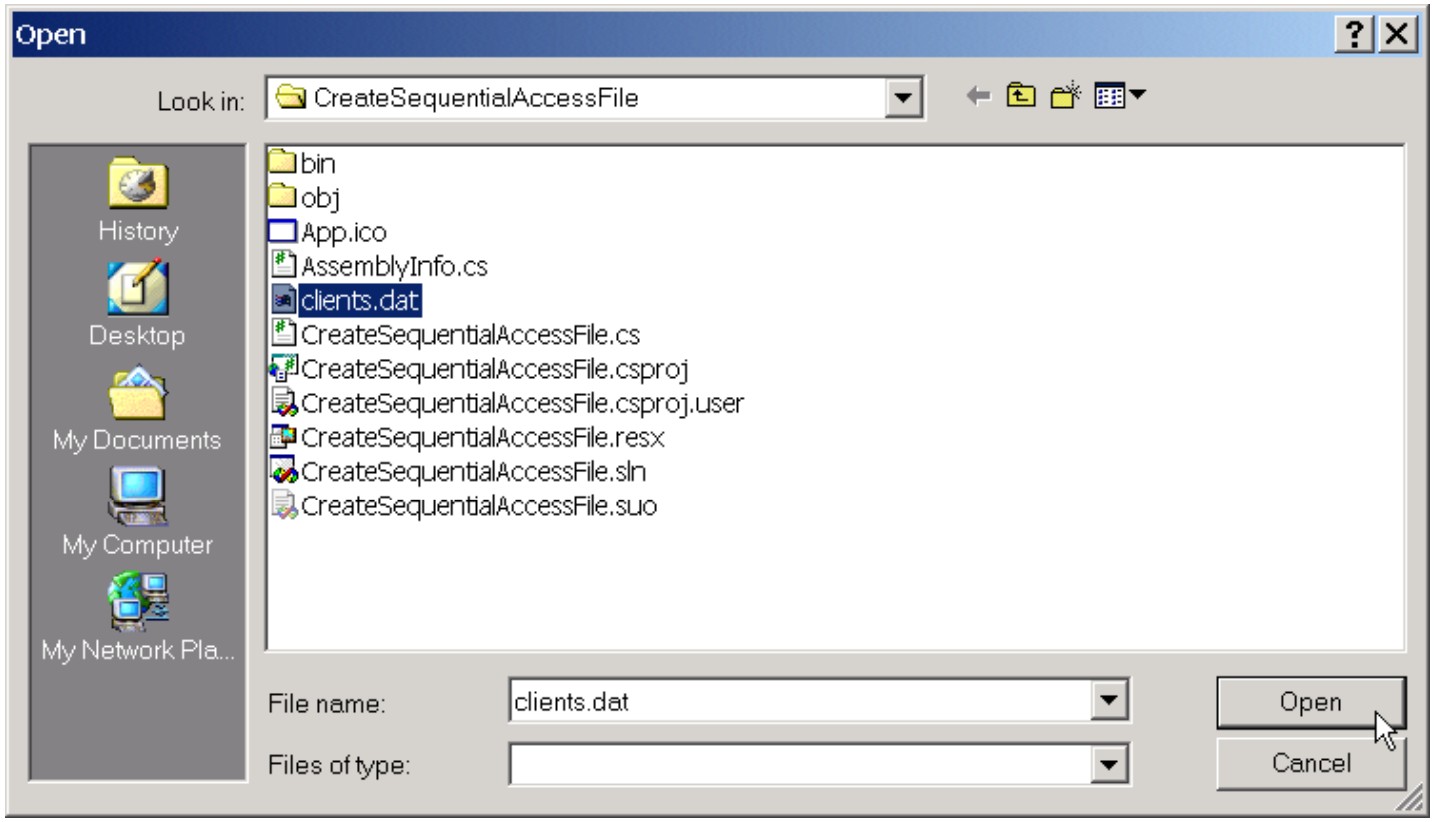
Program Output

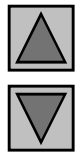


Outline

ReadSequentialAccessFile.cs

Program Output

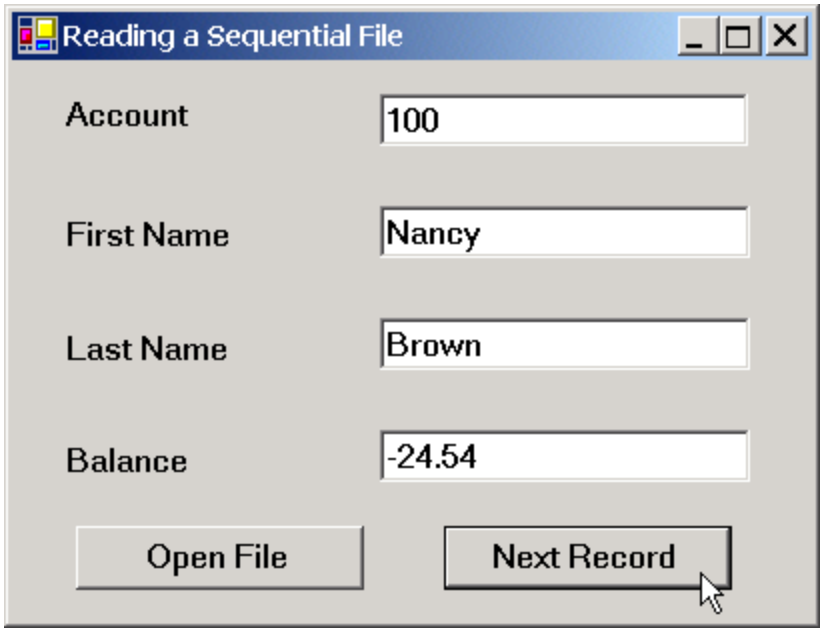


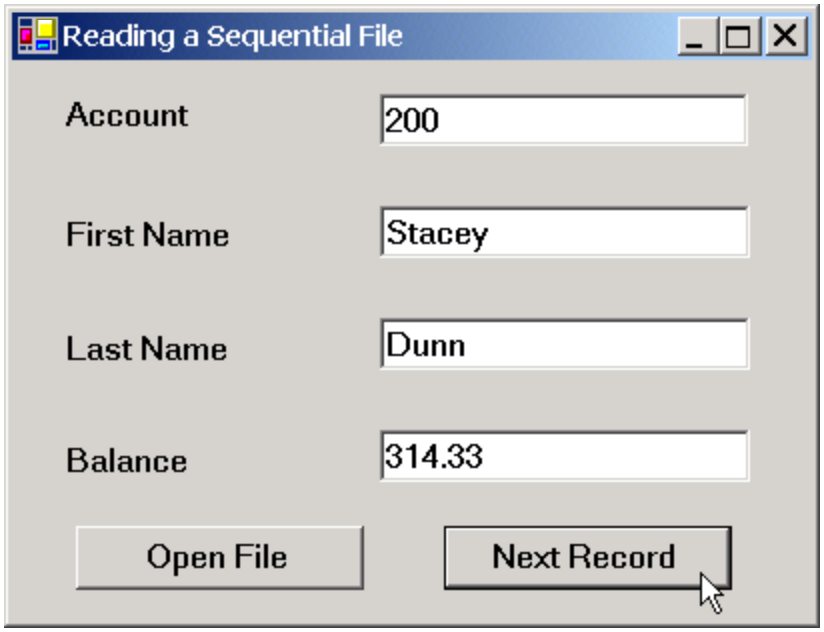


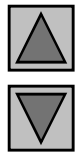
Outline

ReadSequentialAccessFile.cs

Program Output



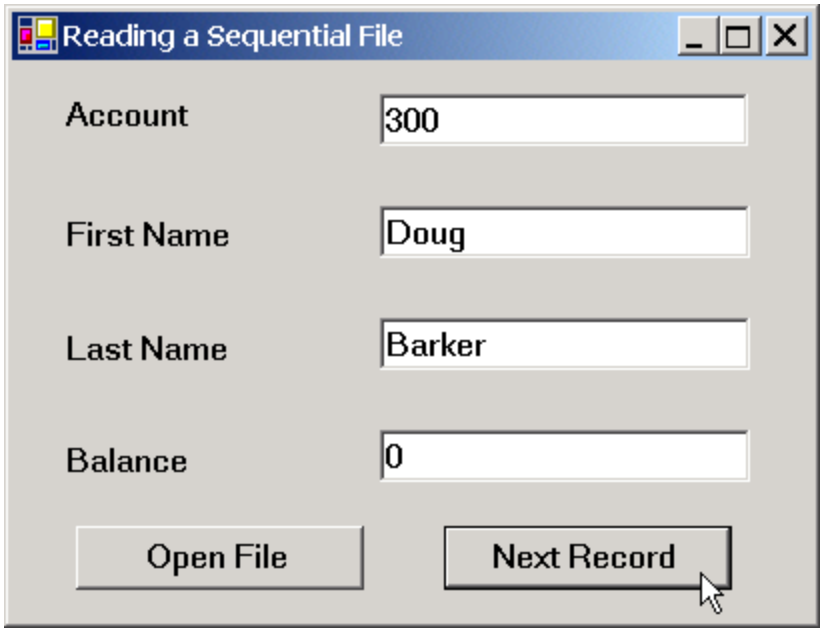


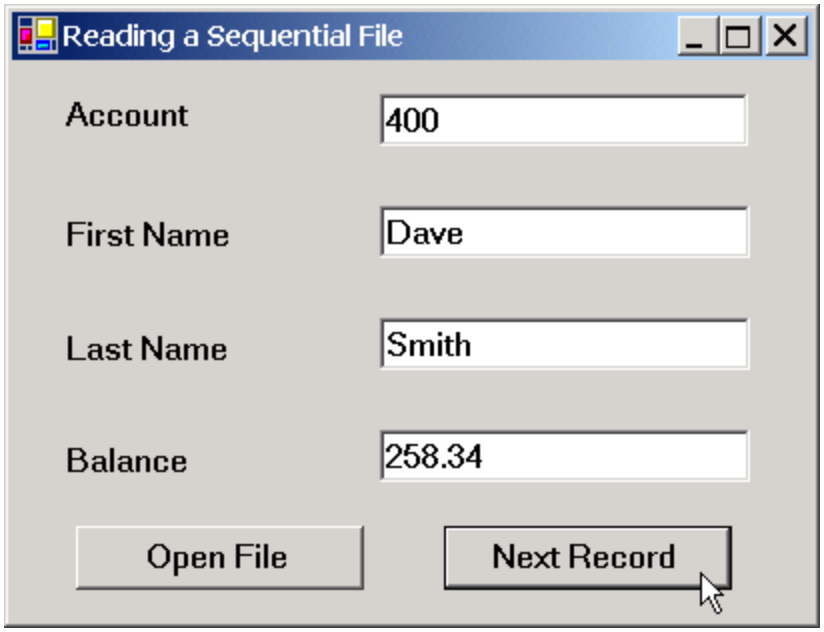


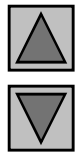
Outline

ReadSequentialAccessFile.cs

Program Output



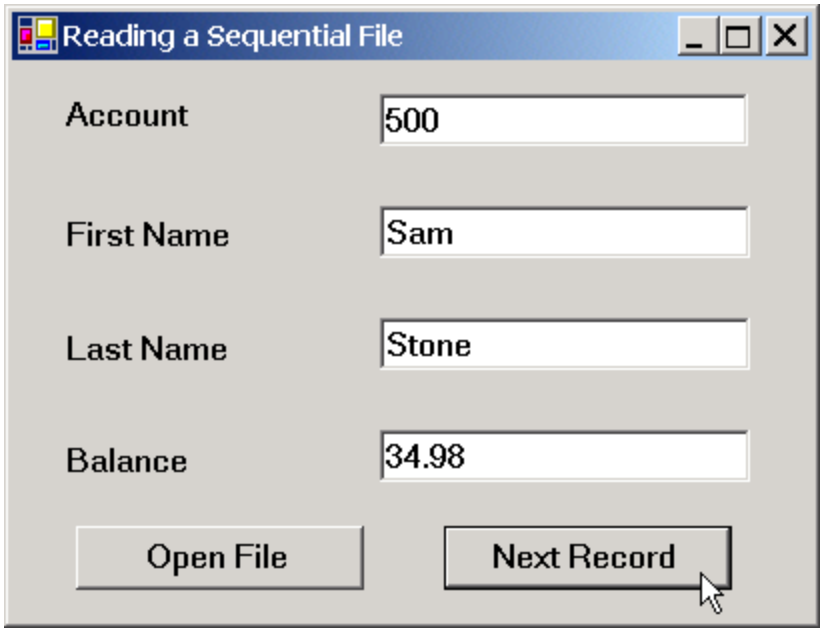




Outline

ReadSequentialAccessFile.cs

Program Output

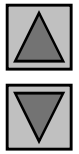


Analyze the following program on your own as a Homework

18.6 Reading Data from a Sequential-Access File

- **`RichTextBox`**:
  - Provides more functionality than **`TextBox`**
  - Among others, provides:
    - **`LoadFile`**: method to display file contents
    - **`Find`**: method for searching individual strings
    - Can display multiple lines of text by default
      - **`TextBox`** by default displays one line of text only

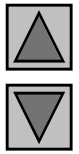
- See input/output behavior of the program: Slides 61 - 62



Outline

CreditInquiry.cs

```
1   // Fig. 17.12: CreditInquiry.cs
2   // Read a file sequentially and display contents based on
3   // account type specified by user (credit, debit or zero balances).
4
5   // C# namespaces
6   using System;
7   using System.Drawing;
8   using System.Collections;
9   using System.ComponentModel;
10  using System.Windows.Forms;
11  using System.Data;
12  using System.IO;
13  using System.Runtime.Serialization.Formatters.Binary;
14  using System.Runtime.Serialization;
15
16  // Deitel namespaces
17  using BankLibrary;
18
19  public class CreditInquiryForm : System.Windows.Forms.Form
20  {
21     private System.Windows.Forms.RichTextBox displayTextBox;
22
23     private System.Windows.Forms.Button doneButton;
24     private System.Windows.Forms.Button zeroButton;
25     private System.Windows.Forms.Button debitButton;
26     private System.Windows.Forms.Button creditButton;
27     private System.Windows.Forms.Button openButton;
28
29     private System.ComponentModel.Container components = null;
30
31     // stream through which serializable data are read from file
32     private FileStream input;
33
34     // object for deserializing Record in binary format
35     BinaryFormatter reader = new BinaryFormatter();
```



Outline

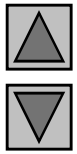
CreditInquiry.cs

```
36
37       // name of file that stores credit, debit and zero balances
38       private string fileName;
39
40       [STAThread]
41       static void Main()
42       {
43          Application.Run( new CreditInquiryForm() );
44       }
45
46       // Visual Studio .NET generated code
47
48       // invoked when user clicks Open File button
49       private void openButton_Click(
50          object sender, System.EventArgs e )
51       {
52          // create dialog box enabling user to open file
53          OpenFileDialog fileChooser = new OpenFileDialog();
54          DialogResult result = fileChooser.ShowDialog();
55
56          // exit event handler if user clicked Cancel
57          if ( result == DialogResult.Cancel )
58             return;
59
60          // get name from user
61          fileName = fileChooser.FileName;
62
63          // show error if user specified invalid file
64          if ( fileName == "" || fileName == null )
65             MessageBox.Show( "Invalid File Name", "Error",
66                MessageBoxButtons.OK, MessageBoxIcon.Error );
```

User clicked open button

Instantiate OpenFilelDialog

Show OpenFileDialog



Outline

CreditInquiry.cs

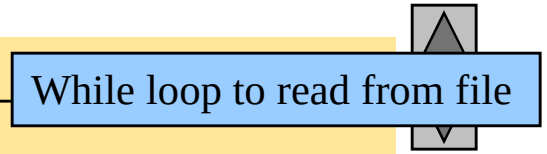
```
67          else
68          {
69             // enable all GUI buttons, except for Open file button
70             openButton.Enabled = false;
71             creditButton.Enabled = true;
72             debitButton.Enabled = true;
73             zeroButton.Enabled = true;
74          }
75
76       } // end method openButton_Click
77
78       // invoked when user clicks credit balances,
79       // debit balances or zero balances button
80       private void get_Click( object sender, System.EventArgs e )
81       {
82          // convert sender explicitly to object of type button
83          Button senderButton = ( Button )sender;
84
85          // get text from clicked Button, which stores account type
86          string accountType = senderButton.Text;
87
88          // read and display file information
89          try
90          {
91             // close file from previous operation
92             if ( input != null )
93                input.Close();
94
95             // create FileStream to obtain read access to file
96             input = new FileStream( fileName, FileMode.Open,
97                FileAccess.Read );
98
99             displayTextBox.Text = "The accounts are:\r\n";
100
```

Button click event handler

Reference to object that sent event

Get text from clicked button

Create read only FileStream for input

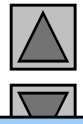


CreditInquiry.cs

While loop to read from file

Read input from file

```
101          // traverse file until end of file
102          while ( true )
103          {
104             // get next Record available in file
105             Record record = ( Record )reader.Deserialize( input );
106
107             // store record's last field in balance
108             Double balance = record.Balance;
109
110             // determine whether to display balance
111             if ( ShouldDisplay( balance, accountType ) )
112             {
113                // display record
114                string output = record.Account + "\t" +
115                   record.FirstName + "\t" + record.LastName +
116                   new string( ' ', 6 ) + "\t";
117
118                // display balance with correct monetary format
119                output += String.Format(
120                   "{0:F}", balance ) + "\r\n";
121
122                // copy output to screen
123                displayTextBox.Text += output;
124             }
125          }
126       }
127
128       // handle exception when file cannot be closed
129       catch( IOException )
130       {
131          MessageBox.Show( "Cannot Close File", "Error",
132             MessageBoxButtons.OK, MessageBoxIcon.Error );
133       }
134
```



Outline

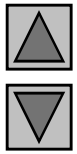
```
135         // handle exception when no more records
136         catch( SerializationException )
137         {
138            // close FileStream if no Records in file
139            input.Close();
140         }
141
142      } // end method get_Click
143
144      // determine whether to display given record
145      private bool ShouldDisplay( double balance, string accountType )
146      {
147         if ( balance > 0 )
148         {
149            // display credit balances
150            if ( accountType == "Credit Balances" )
151               return true;
152         }
153
154         else if ( balance < 0 )
155         {
156            // display debit balances
157            if ( accountType == "Debit Balances" )
158               return true;
159         }
160
161         else // balance == 0
162         {
163            // display zero balances
164            if ( accountType == "Zero Balances" )
165               return true;
166         }
167
```

No more records exception

Close FileStream

Method to determine whether to display each record in file

nquiry.cs



Outline

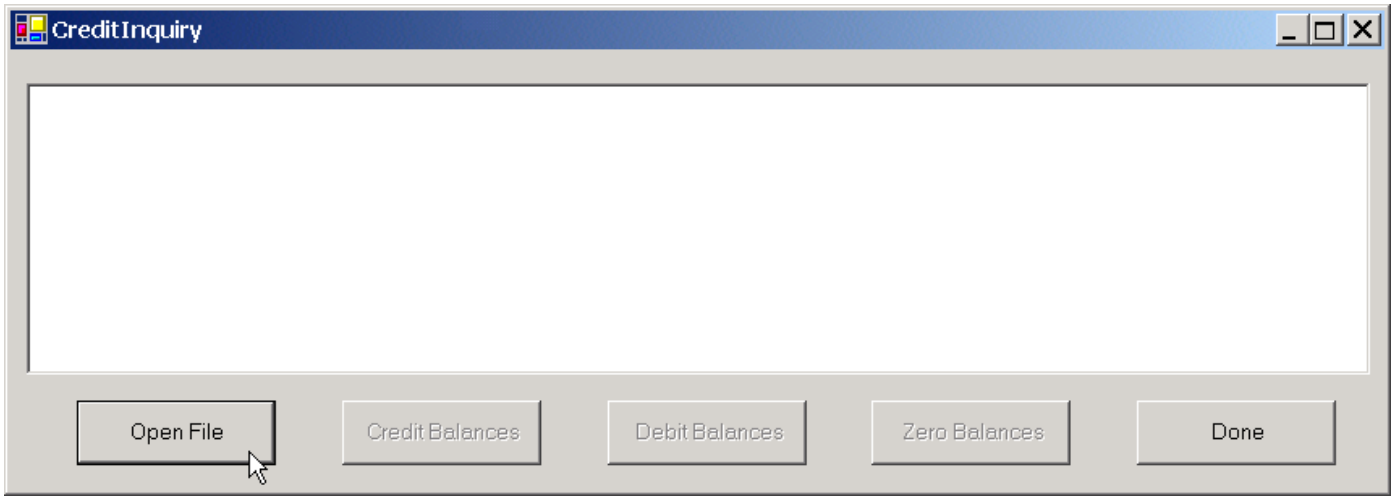
CreditInquiry.cs

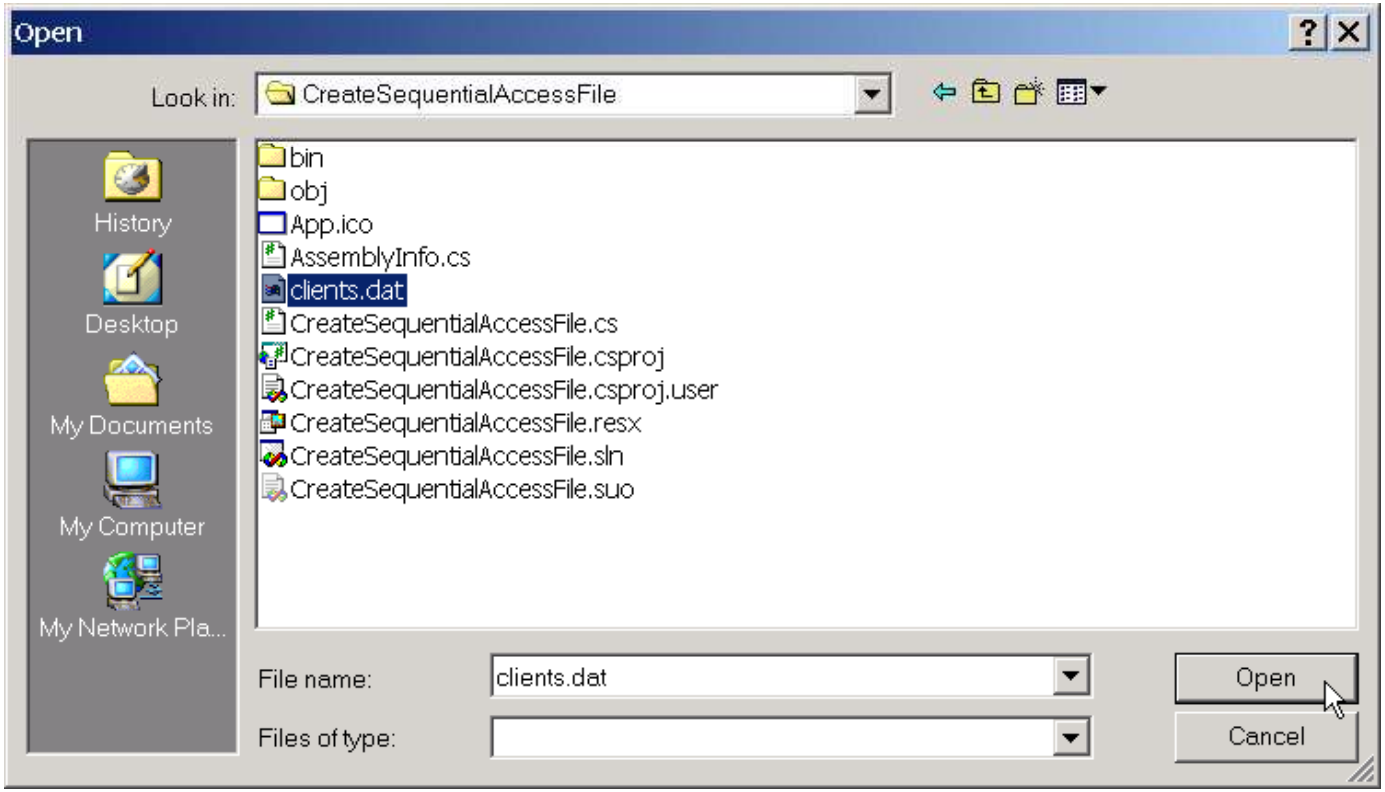
```
168         return false;
169
170      } // end method ShouldDisplay
171
172      // invoked when user clicks Done button
173      private void doneButton_Click(
174         object sender, System.EventArgs e )
175      {
176         // determine whether file exists
177         if ( input != null )
178         {
179            // close file
180            try
181            {
182               input.Close();
183            }
184
185            // handle exception if FileStream does not exist
186            catch( IOException )
187            {
188               // notify user of error closing file
189               MessageBox.Show( "Cannot close file", "Error",
190                  MessageBoxButtons.OK, MessageBoxIcon.Error);
191            }
192         }
193
194         Application.Exit();
195
196      } // end method doneButton_Click
197
198   } // end class CreditInquiryForm
```

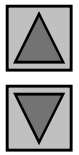
Done button clicked event handler

Outline

`CreditInquiry.cs`
Program Output



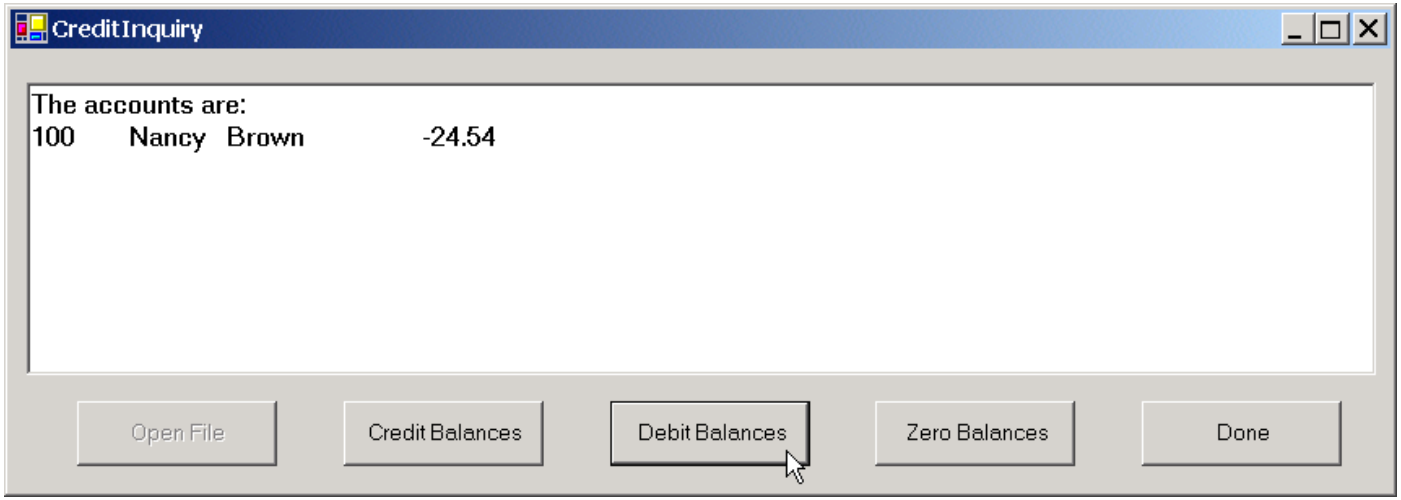


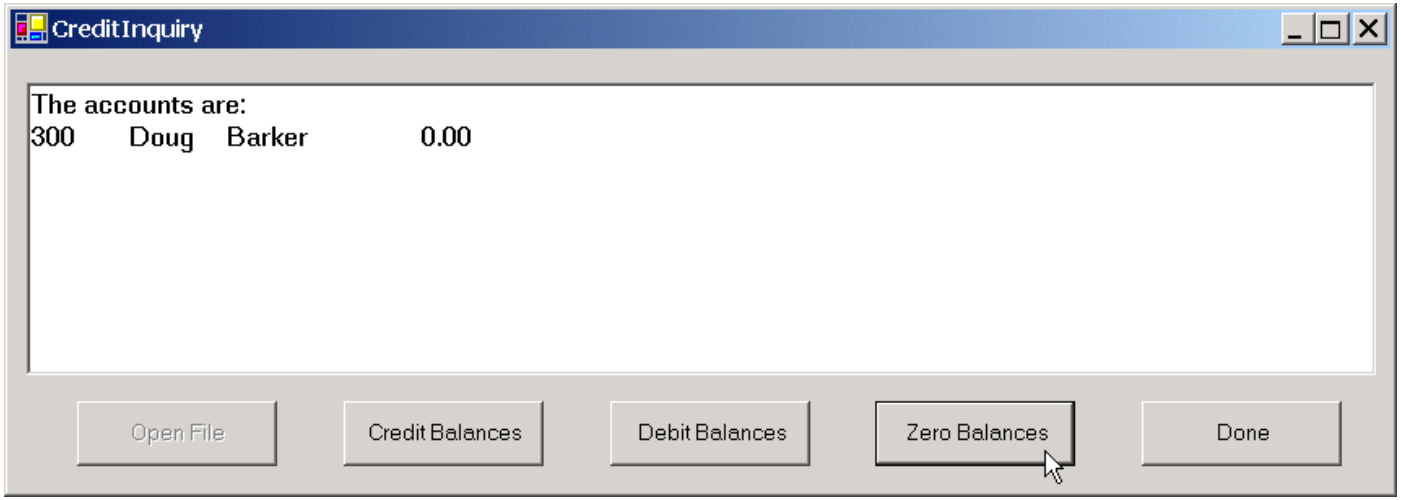


Outline

`CreditInquiry.cs`

Program Output





*** SKIP ***

Section Titles below are from Ed. 1 of the Textbook (hence, from Ch.17)

Topics from 18.7 – 18.9 of Ed. 2 are covered in 18.1 – 18.6 above

17.7 Random-Access Files

17.8 Creating a Random-Access File

17.9 Writing Data Randomly to a Random-Access File

17.10 Reading Data Sequentially from a Random-Access File

17.11 Case Study: A Transaction-Processing Program